2009

Network coding-based survivability techniques for multi-hop wireless networks

Osameh Al-kofahi
Iowa State University

Network coding-based survivability techniques for multi-hop wireless networks

by

Osameh Al-Kofahi

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Ahmed E. Kamal, Major Professor
Daji Qiao
Morris Chang
Aditya Ramamoorthy
Cai Ying

Iowa State University

Ames, Iowa

2009

DEDICATION

To my parents and my wife

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my thanks and gratitude to those who helped me during my study at Iowa state university. I am especially grateful to my advisor Dr. Ahmed Kamal for his guidance, patience and support throughout my research career at ISU. He always directed me in the right track, and offered me valuable advice. I would like to thank him for the efforts and time he spent with me in discussing ideas and writing papers. I honestly consider myself fortunate to have this precious opportunity to work with Dr. Kamal.

I would like to thank my committee members Dr. Daji Qiao, Dr. Morris Chang, Dr. Aditya Ramamoorthy, and Dr. Cai Ying, for their time and feedback. I would also like to thank Dr. Srikanta Tirthapura for his guidance and advice.

I am also greatly thankful to my wife (Sahar), and my parents (Mahmoud and Khitam) for their love, support, and encouragement.

Finally, I thank Yarmouk University for their financial support.

ABSTRACT

Multi-hop Wireless Networks (MWN) have drawn a lot of attention in the last decade, and will continue to be a hot and active research area in the future also. MWNs are attractive because they require much less effort to install and operate (compared to wired networks), and provide the network users with the flexibility and convenience they need. However, with these advantages comes a lot of challenges. In this work, we focus on one important challenge, namely, network survivability or the network ability to sustain failures and recover from service interruption in a timely manner. Survivability mechanisms can be divided into two main categories; Protection and restoration mechanisms. Protection is usually favored over restoration because it usually provides faster recovery. However, the problem with traditional protection schemes is that they are very demanding and consume a lot of network resources. Actually, at least 50% of the used resources in a communication session are wasted in order to provide the destination with redundant information, which can be made use of only when a network failure or information loss occurs. To overcome this problem and to make protection more feasible, we need to reduce the used network resources to provide proactive protection without compromising the recovery speed. To achieve this goal, we propose to use network coding. Basically, network coding allows intermediate network nodes to combine data packets instead of just forwarding them as is, which leads to minimizing the consumed network resources used for protection purposes. In this work we give special attention to the survivability of many-to-one wireless flows, where a set of N sources are sending data units to a common destination T. Examples of such many-to-one flows are found in Wireless Mesh Networks (WMNs) or Wireless Sensor Networks (WSNs). We present two techniques to provide proactive protection to the information flow in such communication networks. First, we present a centralized approach,

for which we derive and prove the sufficient and necessary conditions that allows us to protect the many-to-one information flow against a single link failure using only one additional path. We provide a detailed study of this technique, which covers extensions for more general cases, complexity analysis that proves the NP-completeness of the problem for networks with limited min-cuts, and finally performance evaluation which shows that in the worst case our coding-based protection scheme can reduce the useful information rate by 50% (i.e., will be equivalent to traditional protection schemes). Next, we study the implementation of the previous approach when all network nodes have single transceivers. In this part of our work we first present a greedy scheduling algorithm for the sources transmissions based on digital network coding, and then we show how analog network coding can further enhance the performance of the scheduling algorithm. Our second protection scheme uses deterministic binary network coding in a distributed manner to enhance the resiliency of the Sensors-to-Base information flow against packet loss. We study the coding efficiency issue and introduce the idea of relative indexing to reduce the coding coefficients overhead. Moreover, we show through a simulation study that our approach is highly scalable and performs better as the network size and/or number of sources increases. The final part of this work deals with unicast communication sessions, where a single source node S is transmitting data to a single destination node T through multiple hops. We present a different way to handle the "survivability vs. bandwidth" tradeoff, where we show how to enhance the survivability of the S-T information flow without reducing the maximum achievable S-T information rate. The basic idea is not to protect the bottleneck links in the network, but to try to protect all other links if possible. We divide this problem into two problems: 1) pre-cut protection, which we prove it to be NP-hard, and thus, we present an ILP and a heuristic approach to solve it, and 2) post-cut protection, where we prove that all the data units that are not delivered to T directly after the min-cut can be protected against a single link failure. Using network coding in this problem allows us to maximize the number of protected data units before and after the min-cut.

CHAPTER 1. OVERVIEW

1.1 Introduction

Multihop wireless networks, such as ad-hoc, sensor and mesh networks, have drawn a lot of attention in the last decade, and will continue to be an important research topic in the future also. Applications for these types of networks are numerous and diverse ranging from military to public safety, health and environmental applications. The most important merit of multihop wireless networks, which makes them very attractive is their ease of deployment compared to wired networks that need a pre-installed infrastructure to operate. However, this flexibility compromises the robustness of these networks. For example, the nodes in a sensor or ad-hoc network usually have limited power supply, which causes the nodes to die out and interrupt the network information flow or reduce network connectivity. Moreover, the wireless communication medium is prone to various types of interference and impairments causing a wireless link status to dynamically change according to the channel conditions, and thus causing the wireless links to be intermittently unavailable. Besides interference and impairments, the harsh surrounding environments and severe weather conditions may damage either nodes or links (e.g, a damaged antenna) if the network is deployed outdoors as in the case of sensor and mesh networks. These problems emphasize the need for mechanisms to enhance the network survivability.

Survivability is usually defined as *the capability of a network to deliver data successfully in a timely manner, even in the presence of failures.* Network survivability is important to sustain continuous uninterrupted service for the network users, and is crucial to maintain the quality of this provided service. For example, in a wireless sensor network that is deployed in a battlefield to monitor enemy activities or levels of toxic chemicals, node failures or packet loss may affect

the quality of the monitoring process, and thus the user may not be able to accurately assess the battlefield situation. Therefore, having a survivable communication mechanism will mitigate the effects of node failures, and will provide the user with a more accurate view. Although survivability is defined as a network property, its realization is coupled with a data transfer session. In every session there is a sending side and a receiving side, each of which may consist of one or more network nodes. In general, network survivability methods can be divided into the following categories:

- *Protection mechanisms:* e.g., (31; 32; 22; 16; 33; 34; 18; 35). Protection is usually achieved by using redundant network resources to carry redundant data units. Usually, data units are duplicated and forwarded on multiple paths from the source to the destination. In this case, a data delivery failure occurs and will be detected only if all paths fail. Otherwise, there is no need to detect the failure or retransmit the information. This is called *proactive protection* and is usually referred to as 1+1 protection. An alternative way to provide protection is to divide the paths into two sets, primary and backup, where only the primary path is used to forward data to the destination. A backup path will only be used if the primary path fails. This is called *reactive protection* and is usually referred to as 1:1. Reactive protection can be extended to M:N, where M backup paths are reserved to protect N primary paths. The M backup paths are shared by the N sources, and can be used by any source if a failure occurs on its primary path, which makes this type of protection more efficient in utilizing the network resources. However, reactive protection is slower than proactive protection since a source must detect a failure first, and then switch the data flow to one of the available backup paths. Note that in both proactive and reactive protection all the paths are reserved in advance even if they are not always used as in reactive protection.

  Although reactive protection, as described, is known and viable in wired networks (14), it is not technically accurate to talk about path reservation in wireless networks, since there are no actual physical links that can be reserved. However, a node in a wireless network might learn multiple paths to the destination during the route discovery process

and can use them in a fashion similar to that of reactive protection. That is, all paths are known a priori and will be used as needed, but without being reserved in advance.

- *Restoration mechanisms:* e.g., (36; 13). In restoration mechanisms only a single path is used from a source to a destination, and no backup paths are found in advance. Therefore, restoration mechanisms consume fewer resources than protection mechanisms. However, restoration does not provide recovery at the speed of protection. This is because failures need to be detected first (unlike proactive protection), after that a resource discovery procedure is invoked (unlike protection techniques in general), and finally rerouting is done to find a different route for the data units. Note that the rerouting mechanism here is different from that in reactive protection. In restoration, no information about the available network resource is known to the node that detected the failure, that is why it needs to discover the network resources first to be able to do the rerouting afterwards. However, under protection, multiple paths are computed a priori, and thus, the rerouting mechanism in reactive protection is very simple and is confined to just switching to an available path from the backup set. Finally, it should be noted that restoration is implicitly implemented in all routing protocols in the form of route maintenance mechanisms.

- *Hybrid mechanisms:* e.g., (37; 38; 39). In this case a mix of protection and restoration mechanisms can be used together.

1.2 Survivability issues and challenges

1.2.1 Scalability

The scalability challenge rises mainly in proactive protection mechanisms. This is because survivability is provided through using redundant network resources to forward redundant data units. There are two problems in such schemes. The first one is the problem of wasted resources. For example, to provide survivability against k-1 failures, at least $\frac{k-1}{k}\%$ of the network resources used in the communication session will be wasted to provide the required redundancy. The second one is the problem of the protection overhead. The high overhead produced from

duplication may affect the network performance and lead eventually to congestion, which becomes more notable as the number of protected sessions increases. In other words, traditional proactive protection approaches do not scale well as the number of communication sessions increases.

To mitigate the effects of duplication, erasure codes or network coding can be used. The main advantage of these techniques is that duplication is eliminated, and thus, the overhead is reduced and the useful throughput is increased. These techniques are discussed and compared to traditional proactive protection mechanisms in Section 1.3.

1.2.2 Network connectivity

Network connectivity is defined as the minimum max-flow between any two nodes in the network, which is equivalent to the minimum link cut between any two nodes in the network. The definition can be extended to cover the minimum node cut also. That is, network connectivity is defined as the minimum number of nodes (or links) that when removed (e.g., due to a failure) the network will be divided into two components A and A', such that no node in A is connected to a node in A' and vice versa. Alternatively, a network is said to be connected if there exists a path between any pair of nodes in the network. Furthermore, the definition can be extended to k-connectivity, where a network is said to be k-node (link) connected if there exists k node (link) disjoint paths between any pair of nodes in the network.

Network connectivity is an important network property that directly affects the network survivability. This is because network connectivity is what limits the number of alternative paths that can be found between a pair of nodes. A certain level of network connectivity can be achieved using node deployment algorithms or satisfied through topology control strategies. Wireless sensor networks motivated the development of numerous such algorithms and strategies. This is because, in many scenarios, WSNs are assumed to be deployed in response to certain large-scale events, such as catastrophes, and thus the deployed network must have a certain level of connectivity to guarantee successful data delivery under these conditions.

1.2.3 Disjoint Vs. Interleaving paths

Multipath routing is the mainstream approach to proactive protection mechanisms. In multipath routing k paths are found between a source node (S) and a destination node (T), these paths can be either node or edge disjoint, or they can be interleaving, i.e., some edges are shared. When a data unit is to be sent from the source to the destination, the source sends k copies of the data unit to the destination on the k paths. If the paths are disjoint, each of which forwards a single copy to T. This guarantees successful data delivery if failures take place on at most k-1 paths out of the disjoint k paths. However, all the copies will be lost if failures occur on all k paths.

If the paths are interleaving, a shared link does not forward all the copies from all the paths, it only carries one of them and the head node of the shared link duplicates the data unit on all of the outgoing paths. In the interleaving paths case, successful data delivery is guaranteed if failures take place on at most p-1 paths, where p is the maximum number of disjoint paths from the k S-T paths, and $p < k$ (if p=k the paths are disjoint). Unlike the disjoint case, interleaving may enhance the chances of the information to reach the destination even if failures occur on all k paths. This is illustrated in Figure 1.1. **In the graph four paths are found** from the source to destination, namely, $P_1 : S \rightarrow A \rightarrow B \rightarrow T$, $P_2 : S \rightarrow A \rightarrow C \rightarrow E \rightarrow T$, $P_3 : S \rightarrow D \rightarrow C \rightarrow B \rightarrow T$, and $P_4 : S \rightarrow D \rightarrow E \rightarrow T$. Note that the choice of the paths is not unique, and other interleaving paths can be chosen. In the 4 paths above, P_1 shares link (S,A) with P_2, and link (B,T) with P_3. P_4 shares link (S,D) with P_3, and link (E,T) with P_2. To see the advantage of interleaving, assume that links (A,B), (A,C), (D,E) and (C,B) have failed. In this case node C will still receive a copy from D, and will send it to node E, which in turn will relay it to the destination.

1.2.4 End-to-End Vs Local Recovery

The recovery process is initiated once a failure is detected. Recovery can be done on an end-to-end basis or it can be done locally. In end-to-end recovery a node that detects a failure notifies the source by sending a specific message. Upon receiving this notification, the source is

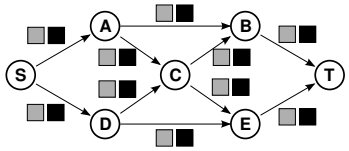


Figure 1.1 4 interleaving paths, and at most 2 are disjoint, i.e., $k = 4$ and $p = 2$

responsible for finding an alternative path to the destination. On the contrary, local recovery is initiated directly at the intermediate node that first detects the failure. In both cases, the alternative path might be stored in the source (or intermediate node) buffer already, or yet, needs to be discovered. This depends on the memory allocation for routing information at each node.

The main advantage of end-to-end recovery is that it provides the best (i.e., least cost) alternative S-T path, since the scope of the search for a new path is from the source to the destination. However, in end-to-end recovery, the recovery time is longer (compared to local recovery) and the wasted bandwidth is more, since the notification message must be forwarded by all the intermediate nodes on the path all the way back to the source. In contrast, local recovery may provide sub-optimal alternative routes (optimal from the detecting node to the destination), but is faster and more efficient. In some cases both techniques are used. Local recovery can be used as a first aid to help packets in transit to reach the destination instead of dropping them, until a new end-to-end path is found and used by the source.

1.3 Network coding: overview and motivation

Network coding was introduced in (26). Instead of just forwarding (routing) packets as is, network coding allows the nodes in a network to create combinations from packets arriving on their incoming links and then forward these combinations on their outgoing links. It was shown in (26) that by using network coding in a multicast connection, one can achieve the multicast capacity, which is defined as the least of the minimum cuts between the multicast source and each of the destinations. An example is shown in Figure 1.2. In the figure, node A

is a multicast source that has the packets b_1 and b_2, and nodes E and G are two destinations, each of which is interested in receiving both b_1 and b_2. Without network coding link (C, F) will be the bottleneck link and will have to alternate between sending b_1 and b_2. However, if we use network coding and allow node C to combine b_1 and b_2, the multicast capacity will be achieved and link (C, F) will not be a bottleneck. This is illustrated in Figure 1.3, where the addition here is modulo 2 (i.e., equivalent to bitwise XOR). In this case node E recovers b_1 by XORing b_2 with $b_1 + b_2$, and similarly G recovers b_2 by XORing b_1 with $b_1 + b_2$.

The theory of network coding was further investigated afterwards. Linear network coding was shown to be sufficient to achieve the multicast capacity in (5). An algebraic framework for network coding was proposed in (6). In (7) the authors took a combinatorial approach and proposed a method to identify structural properties of multicast network configurations. It was shown that different networks may be equivalent from a coding point of view, and thus multicast networks can be sorted into equivalence classes. Polynomial time algorithms were proposed in (8) to assign coding vectors to network links, i.e., to define the relationship between the symbol carried on a link with the set of symbols arriving at the tail node of that link. Besides the work done on the theoretical part of network coding, there were many other works that considered utilizing network coding in a variety of applications. For a review of some of the interesting applications of network coding in wireless networks the reader is referred to (9).

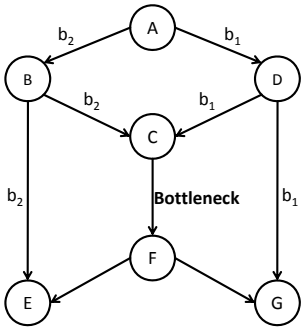


Figure 1.2 Link (C, F) is a bottleneck

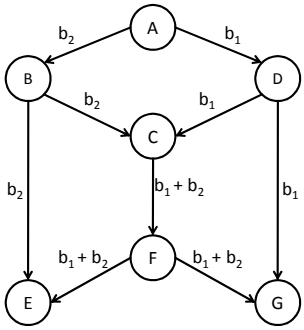


Figure 1.3 Multicast capacity achieved

We now give a simple example to show the advantage of network coding over just routing

in providing proactive protection. Consider the example in Figure 1.4(a), where there are three paths between S and T. In this case, using duplication (i.e., sending two copies of the same data unit to the destination) we cannot protect more than one data unit, since to protect two data units we need four disjoint paths. However, if network coding is allowed, as shown in Figure 1.4(b), the source can send two different (not duplicates) data units to two neighboring nodes on two disjoint paths; d_1 to node A and d_2 to node C. Because of the wireless multicast advantage, node E hears both transmissions and produces the combination $d_1 \oplus d_2$ (bitwise XOR), which can be forwarded to the destination on the third path. This way, the destination receives 3 equations in two unknowns, where any 2 equations are solvable and are enough to recover the original data units. That is, 2 data units are proactively protected against a single node or link failure using only 3 paths, i.e., using 25% fewer paths than duplication. Note that the savings increase as the number of available paths increases.

Figure 1.4 Three node-disjoint paths are available between S and T. (a)Using just routing we cannot protect more than one data unit. (b)Using network coding we can protect two data units.

In addition to network coding, erasure codes can be used to reduce the duplication overhead. In erasure codes each data unit is encoded into n smaller sub-packets. Then, from these n sub-packets, e redundant sub-packets are created. The resulting $n+e$ sub-packets can be forwarded on $n+e$ disjoint paths to the destination. These $n+e$ paths can be either node or link disjoint; it depends on whether we want to tolerate node or link failures. If an optimal erasure code is used, it is enough for the destination to receive n out of these sub-packets to recover the original data unit, i.e., e failures can be tolerated. We now compare duplication to coding based

approaches, i.e., both erasure codes and network coding. We assume a simple topology, similar to that in Figure 1.4, where we have a source/destination pair S and T, with k node-disjoint S-T paths in between. To simplify the comparison, we assume that k is an even number, and that all the paths have the same number of hops, L. In addition, we assume that protection is to be provided for the S-T information flow against a single failure, and that the one hop propagation delay is τ. We compare these different techniques based on the useful throughput. In our computations we take the transmission conflicts between the nodes along the same path into account, but we ignore the conflicts between nodes on different paths for simplicity. We assume that the interference range equals the transmission range, i.e., a node can only transmit to and interfere with all its 1-hop neighbors on the same path.

To tolerate a single failure in duplication-based approaches, each packet is forwarded on 2 disjoint paths. Therefore, if there are k paths the source can use a different pair of paths for each packet to distribute power consumption. Since a node cannot transmit and receive at the same time, the source can transmit a packet every 3τ because it needs to wait for the 2-hop neighbors to transmit first so that their transmissions do not conflict with its transmission to the 1-hop neighbors. Therefore, the rate of receiving useful information at the destination is $1/3\tau$ (i.e., one data unit every 3τ). Note that the throughput is independent of the number of paths.

In erasure codes, each sub-packet is transmitted alone and needs roughly τ/n time units (since its size is smaller). We assume an optimal erasure code in which only a single redundant sub-packet is generated, i.e., a total of $n+1$ sub-packets are transmitted to provide protection against a single failure. In this case, the source can transmit the $n+1$ sub-packets every $(n+1+2)\tau/n = (n+3)\tau/n$ time units. Therefore, the rate in this case is $n/\tau(n+3)$. Obviously, the rate is a function of n. However, it also depends on the number of paths indirectly, since $n+1$ cannot be larger than k.

When network coding is used, the k paths carry k combinations in k-1 data units to the destination, such that any k-1 of them are solvable. This can be easily accomplished by sending k-1 native (uncoded) data units to k-1 first-hop neighbors of the source. Because of the wireless

multicast advantage, the last 1-hop neighbor will be able to overhear these k-1 transmissions and XOR all the received data units to create the last combination. Since only k-1 packets are transmitted (not sub-packets), only $(k-1+2)\tau$ time units are needed. Therefore, the useful data rate at the destination in this case is $(k-1)/\tau(k+1)$. Note that the rate depends clearly on the number of paths. Figure 1.5 plots the rate for the three cases, where the x axis is the number of sub-packets n. Note that the performance of duplication and network coding is independent of n. The dashed Gray line is the rate for duplication, which is a constant and independent of n. The Gray dotted line represents the rate for erasure codes, which clearly gets better as the number of sub-packets increases. The erasure codes rate is drawn for the case when k = 10. However, for smaller values of k the rate follows the same trend but the function will be undefined after n=k-1, since the total number of sub-packets (n+1) cannot exceed the number of paths (k). The set of Black lines (dotted, dashed and solid), represent the rate for network coding, where each line represents the rate for a certain k. As in erasure codes, the lines representing the network coding performance for some k cannot extend beyond n=k-1. Note that when an erasure code is used, in each transmission round the source makes k short transmissions, one for each sub-packet. That is, to transmit k-1 packets the source needs $(k \times \tau/n)(k-1)$ time units, and since $n = k-1$ the source will eventually need $k\tau$ time units. However, when network coding is used only $(k-1)\tau$ time units are needed, which establishes the difference in performance between erasure codes and network coding.

1.4 Contributions

Our work focuses on developing network coding-based protection techniques, which are more feasible and efficient in utilizing the available network resources, when compared to traditional protection mechanisms. We give special attention to the survivability of wireless many-to-one flows, since this type of communication is dominant in two important and highly evolving networks, namely, Wireless Mesh Networks (WMNs) and Wireless Sensor Networks (WSNs). We also study and present a new method to enhance the survivability of the information flow between a given pair of nodes, in an ad hoc or wireless mesh network, that does not

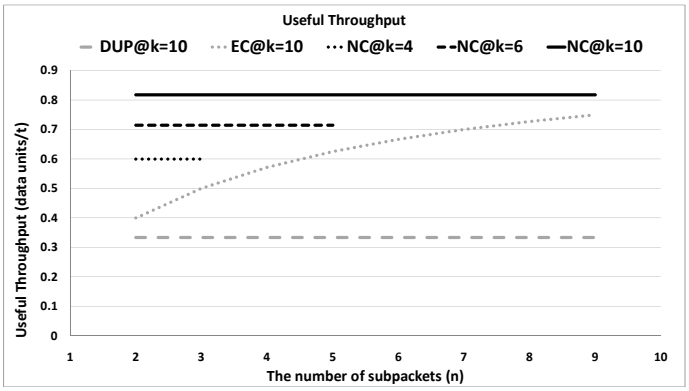


Figure 1.5 Useful throughput of the three schemes in (data units/τ). The useful duplication rate is constant. The erasure codes rate is dependent on n. Finally, the network coding case is dependent on the number of paths k, and achieves better rate compared to duplication and erasure codes

reduce the maximum achievable rate between these nodes. Our contributions are summarized in the following points:

1. A new network coding-based technique is proposed to provide protection for many-to-one flows in WMNs.

    - We derive and prove the necessary and sufficient conditions to provide network coding-based protection for such many-to-one flows against single and multiple failures.
    - We present the coding tree algorithm, which is a simple coding algorithm that uses binary network coding. The coding tree combines the data units from N sources to produce $N + 1$ combinations, such that any N of them are solvable.
    - Our solution is extended for networks with limited min-cuts (when the min-cut is less than $N + 1$). In this case, we show that the problem becomes NP-complete.
    - We study the problem of scheduling the sources transmissions based on both digital and analog network coding. We show that by using analog network coding the number of time slots of a transmission schedule based on digital network coding can

be reduced by a factor of at most N theoretically. Our simulations show that the number of time slots is usually reduced only to half.

2. We present a new distributed network coding-based technique that enhances the survivability of the many-to-one flow in a WSN against packet loss.

   - We use binary network coding in a distributed fashion, where each source node makes the coding decisions independently from other nodes.
   - The technique allows N sources to collaborate in a simple distributed manner to produce $N + 1$ combinations such that any N of them are solvable.
   - The solution is extended to cover multiple packet losses. In addition, a simple routing protocol is presented that allows us to use this scheme to tolerate node and link failures also.
   - Relative indexing is introduced to reduce the coding vectors overhead.

3. We introduce the concept of *Max-flow protection*, which can enhance the survivability of the information flow between a pair of nodes, S and T, without reducing the maximum achievable S-T max-flow.

   - We present the idea of extra connectivity with respect to a given S-T max-flow, where we define extra source and destination connectivity.
   - We study the problem of pre-cut protection, and show that it is an NP-hard problem. Therefore, we formulate it as an ILP and we also present a heuristic approach to solve it.
   - We also study the problem of post-cut protection, and prove that all data units, which are not delivered directly to T after the min-cut, can be post-cut-protected against at least a single failure if network coding is allowed.

1.5 Organization

The rest of this thesis is organized as follows. In Chapter 2 we review and discuss some of the most well known survivability mechanisms for wireless networks proposed in the literature. The first part of our work is presented in Chapter 3, where we present a centralized network coding-based protection scheme for many-to-one wireless flows. In Chapter 4, we augment the work in previous chapter, where we study the problem of scheduling the sources transmissions based on both digital network coding and analog network coding. In Chapter 5 we present a simple technique that uses deterministic binary network coding in a distributed manner to enhance the survivability of the sensors-to-base information flow against packet loss. In Chapter 6, we introduce the concept of *max-flow protection* in a communication session between a single source S and a single destination T. Max-flow protection allows us to enhance the survivability of the S-T information flow without reducing the maximum achievable S-T information rate. Finally we conclude the thesis in Chapter 7

CHAPTER 2. Literature Review

In this chapter we review and discuss some of the most well known survivability mechanisms for wireless networks proposed in the literature. This discussion is by no means exhaustive, but the discussed mechanisms are sampled in a way that covers the whole spectrum of the survivability approaches. We first summarize the types of failures in multihop wireless networks, which can be classified into the following types:

1. Node failures
2. Link failures
3. Service node failures (such as access points, gateways, base stations, or cluster heads)

We distinguish between failures of normal and service nodes because the failure of a service node has a larger impact on the network since all the nodes associated with it are affected. It should be noted that we focus on the survivability mechanisms that enhance the survivability of communication sessions, and not on the mechanisms that enhance the survivability of individual network components. In other words, we focus on mechanisms that mitigate the effects and not the causes of failures.

2.1 Proactive Protection Mechanisms

The most agile class of survivability methods is proactive protection, since redundancy (in information and used resources) ensures that the destination will receive the information even if a failure occurs. Because of this fact, most of the previous work in the survivability of multihop wireless networks belongs to this category. Approaches to solve the problem of finding multiple disjoint paths can be theoretical (based on graph theory or network flows concepts)

or practical in the form of protocols. The authors in (31) take an algorithmic approach, and introduce centralized optimal polynomial-time algorithms for finding either minimum energy link-disjoint or minimum energy node-disjoint paths in wireless Ad Hoc Networks. The proposed algorithms use known minimum-weight k-disjoint paths algorithms (e.g, Bhandari's or SuurballeŠs algorithms) on a transformed graph. The transformation takes any graph G, and transforms it to a fully connected graph G' (i.e., with $\binom{n}{2}$ links), where each link is assigned a cost that represents the needed power to transmit on it by any end node. The algorithms minimize the total energy on all the used paths by exploiting the wireless multicast advantage (WMA), which also makes them more suitable for wireless networks. Specifically, for the node-disjoint case the problem reduces to optimizing the transmission energy at the source, so that its transmission can reach a suitable set of neighbors that allows establishing k node-disjoint S-T paths. For the link-disjoint case, the problem reduces to finding node-disjoint paths between common nodes on the link disjoint paths using the previous algorithm. The proposed algorithms optimally solves the 2 link-disjoint paths problem in $O(kN^5)$, and the k node-disjoint paths problem in $O(kN^3)$.

On demand routing protocols that are able to find multiple node or link disjoint paths, between a source and destination pair, were developed for ad-hoc networks. Some of these routing protocols are extensions to well known routing protocols such as Ad-hoc On-demand Distance Vector routing (AODV), and Dynamic Source Routing (DSR). There are two main differences between single-path and multi-path routing protocols. First, intermediate nodes are allowed to forward duplicate RREQ (Route Request) messages to give the destination more options to choose from. Duplicate RREQ messages result from broadcasting the RREQ by the original source and all the nodes that hear it afterwards. Second, intermediate nodes are not allowed to reply if they have a valid route to the destination in their routing table, in order to ensure disjointedness between paths.

In (25) AOMDV (Ad-hoc On-demand Multipath Distance Vector routing) was introduced as an extension to AODV to calculate multiple link-disjoint paths. Similar to AODV, AOMDV uses RREQ and RREP messages to discover routes. Specifically, a new field is added to the

RREQ message which identifies the *first-hop*. If multiple RREQ messages are received only the first is forwarded (similar to AODV). Upon receiving multiple RREQs an intermediate node can discover multiple node-disjoint paths to the source if they have different first-hops. The destination replies to k of the RREQs regardless of their first-hop, where each of which is sent to a different neighbor. Upon receiving multiple RREP messages, an intermediate node sends each one on a different path to the source so that multiple link-disjoint paths can be created. This is illustrated in Figure 2.1.

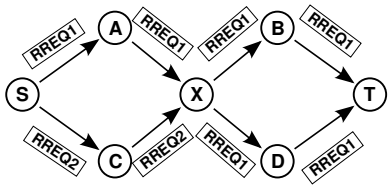


Figure 2.1 AOMDV: RREQ1 and RREQ2 are copies of the same RREQ broadcasted by S. At node X, the copy RREQ2 is discarded, and X knows 2 node-disjoint paths to the source. Although not shown, the destination sends k RREPs to k different neighbors, in this example k=2. Similarly, if an intermediate node receives duplicate RREPs it forwards each one to a different neighbor, and thus k link-disjoint paths are created

AODVM (AODV-Multipath), another extension to AODV, was proposed in (24) to find node-disjoint paths. In this approach an intermediate node forwards all the RREQs it receives, and keeps a table (called the "RREQ Table") in which it records all the neighbors from which it received the RREQs. Intermediate nodes are not allowed to send back RREP messages. Therefore, RREPs are sent only from the destination node, where for each RREP message the destination includes a new field that contains the ID of the last-hop to the destination (to distinguish node-disjoint paths). Upon hearing an RREP from a neighbor, an intermediate node deletes the neighbor's entry in its RREQ table (if there is any) and inserts a new route in its routing table. In addition, if a node overhears an RREP message from a neighbor it also deletes the neighbor's entry in its RREQ table to prevent a node from participating in multiple paths (to guarantee node disjointedness). The authors propose using reliable nodes (or R-nodes for short) to increase the number of reliable paths, where it is assumed that R-nodes

do not fail at any time. A reliable path is composed of a set of connected reliable segments, where a reliable segment, in turn, is defined either as a set of k disjoint paths between two reliable nodes (k is a design parameter) or a path composed only from reliable nodes. This is clarified in Figure 2.2. The authors show that randomly placing the R-nodes in the network does not increase the number of reliable path a lot, and thus they propose a placement strategy that relies on the randomized min-cut algorithm. They assume that a node knows the local network topology up to a certain number of hops. From this knowledge each node calculates the min-cut in this local sub-graph using the randomized min-cut algorithm. This information is then spread using HELLO messages that are also used to discover the topology. R-nodes are placed or make their movement decisions according to the received min-cut information, where an R-node moves to the proximity of a node with the least local min-cut. If two or more nodes have the same local min-cut an R-node moves to the proximity of the node with the largest min-cut set (the partition resulting from the cut, that has the largest number of nodes).

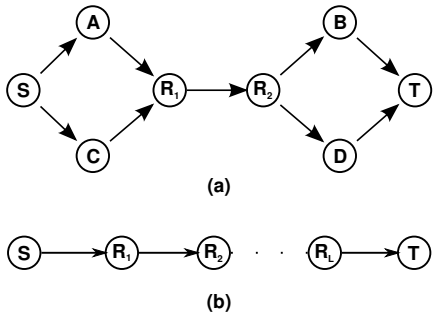


Figure 2.2 AODVM:(a) A path composed of three reliable segments, where k=2. (b) A path composed totally from R-nodes

An extension to DSR, referred to as MP-DSR (Multipath-Dynamic Source Routing), was proposed in (40). MP-DSR is a modification to DSR that enables the computation of multiple node-disjoint paths. In MP-DSR, reliability is treated as a QoS metric, which is used to determine the number of paths to be used. In other words, the reliability of the set of disjoint paths that will be computed by the destination should collectively satisfy a certain reliability requirement. The source starts by determining 1) the lowest acceptable path reliability, $\prod_{low}$, 2) the number of paths, m_0, that the source aims to discover, and 3) the period of time in which

the routes will be used, t_w. The values of $\prod_{low}$ and m_0 are carried in the RREQ messages. In addition, the RREQ messages have a field that contains the accumulated reliability of the traversed path. Upon receiving an RREQ an intermediate node updates the accumulated reliability field in the RREQ, and decides to forward the message if the accumulated reliability is larger than $\prod_{low}$. Each intermediate node is allowed to forward m_0 duplicate RREQ messages. Before sending the RREP messages either 1) the destination node waits a certain time period before running a path selection algorithm to choose the disjoint routes (to have enough options), or 2) waits until the received RREQs give enough paths to satisfy the reliability requirement. Route maintenance is needed only when all paths are broken or when t_w expires.

The routing protocols discussed above are proposed to protect unicast connections. However, other connection structures were also considered. For example in (32) to achieve survivable broadcast and multicast, the use of redundant trees was proposed. Basically two broadcast/multicast trees are created, and then, information is forwarded on both trees. The two trees are said to be survivable, if for every destination node each tree has a path from the source to that node, such that the two paths are node-disjoint. Two flavors of this problem were introduced;1) min-max survivable broadcast/multicast trees, in which the maximum used power by any node is minimized and 2) minimum survivable broadcast/multicast trees in which the sum of the transmission power of all the nodes is minimized. An optimal algorithm was presented for the first problem of order $O(n^2 logn)$ and an effective heuristic was given for the second problem of order $O(n^2(m+n))$.

In addition to Ad-hoc networks, many survivable routing protocols have been developed for WSNs. To make use of the dense deployment of WSNs, the authors in (16) presented GRAdient Broadcast (GRAB). Basically, a cost field is first constructed, which assigns each node a cost that represents the needed energy to forward a packet from this node to the sink along the least cost path. Then, when an event occurs, the sensors in the proximity of the event elects a node called the Center of Stimulus (CoS), which has the best reading and will be the only node to send a report to the BS. The CoS assigns a credit $(\alpha + C_{CoS})$ to each report it creates, where C_{CoS} is the cost of the CoS, and α is an additional credit calculated by the

CoS. Upon receiving a report, an intermediate node checks the remaining credit in the report, if the ratio of the remaining credit to the original credit is higher than the ratio of the current node cost ($C_{current}$) to the original source cost (C_{source}), i.e., $\frac{\alpha-\alpha_{used}}{\alpha} \geq (\frac{C_{current}}{C_{source}})^2$, the node broadcasts the report with a power high enough to guarantee that the nearest 3 downstream neighbors will receive the report. Otherwise, the node uses its minimum cost path to the sink. This creates a forwarding mesh that is composed of a set of interleaving paths, which will forward the report to the sink, as shown in Figure 2.3. Obviously, $\alpha + C_{CoS}$ controls the width of the forwarding mesh; a larger α means more robustness. The authors showed through simulation that when $\alpha \geq 6 * C_{CoS}$ the delivery ratio is more than 95%.

Another work that considered interleaving paths is the one presented in (17). In this paper the authors propose two simple extensions to directed diffusion, to allow the localized computation of multiple node-disjoint paths and multiple braided (interleaving) paths. Two failure modes were defined; 1) patterned, where all the sensor nodes within a circle of some area fail, and 2) isolated, where each node fails independently from the others. The authors compared disjoint-multipath routing to braided-multipath routing using simulation. It was shown that A set of braided paths, which provides a comparable level of survivability of a set of disjoint paths against patterned failures, produces 33% of the overhead of disjoint paths, and provides 50% more resilience to isolated failures.

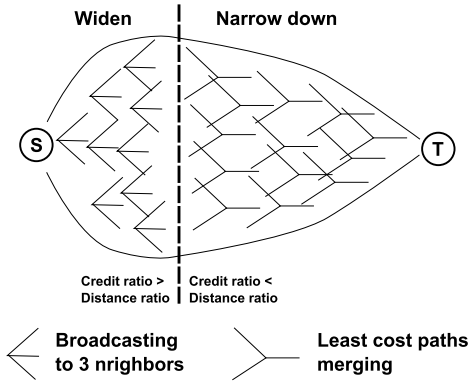


Figure 2.3 GRAB: As long as the remaining credit ratio is larger than the remaining distance ratio, the forwarding mesh width increases. When it is less, each node uses its least cost path to the sink, and these paths might start merging as they approach the sink

Angle-based dynamic path construction was introduced in (41) for WSNs. The mechanism is a variation of geographical routing. In geographical routing a node chooses to forward the data packets to the neighbor that makes the best progress towards the sink, which creates a single path to the sink. However, in angle-based routing, a source node (the one that generates a data packet, not the one that forwards) chooses all its neighbors that are located in a certain area to forward its packet. This area is called the angle zone, which in turn depends on an angle θ_{v_i} called the routing angle. Other sensors that forward the packet perform normal geographical routing. The routing angle depends on the distance from a sensor node to the base station, where the longer the distance the narrower the angle. In order to guarantee that power consumption is distributed among the sensor nodes, it is assumed that a sensor node will notify its neighbors when its remaining energy drops below a certain threshold. Upon receiving a notification, an upstream sensor chooses an alternative downstream node.

To tolerate base station (BS) failures in addition to normal sensor node failures, an algorithm to solve the colored tree multiple pair (CTMP) problem was introduced in (33). Basically, to tolerate BS failures, it was assumed that a WSN may contain more than one base station, and the crux of the algorithm is to find for each node in the network two node-disjoint trees, such that each one of them is rooted at a different BS. Therefore, the WSN can tolerate a single node failure even if it was the BS without loss of information.

2.2 Reactive Protection Mechanisms

To reduce the amount of traffic produced in a proactively protected communication session, and hence energy consumption, reactive protection can be used. In reactive protection mechanisms, multiple paths are known in advance before the communication session is started. However, they are not used unless a failure was detected on the primary path. Split Multipath Routing (SMR) (34) is an example of such reactive protocols. As in other on-demand source routing protocols, the route discovery is initiated at the source by flooding an RREQ message in the network. When duplicate RREQ messages are received by intermediate nodes only those coming from different links (i.e., neighboring nodes) with the number of hops less than that

in the first received RREQ are forwarded, otherwise the message will be discarded (compared to discarding all duplicates as in single path protocols). The reason for this is to give the destination more options to pick maximally disjoint paths. The destination will always choose the path with the least delay (the one included in the first received RREQ) as the primary path. After that, the destination finds the maximally disjoint path(s) with the least delay path (i.e., with the minimum number of shared hops). Then, the path with the least number of hops from those maximally disjoint is selected as an alternative. The authors tested two variations of SMR. In the first, SMR-1, the route discovery process is repeated upon a single failure on any of the paths. In the second, SMR-2, the route discovery process is initialized only when both paths are disconnected. It was shown through simulation that SMR-2 performs better than SMR-1 and outperforms DSR (in terms of packet delivery ratio, delay and overhead).

The many-to-one communication paradigm in wireless sensor networks was considered in (18). An efficient algorithm was proposed to provide each node in the WSN with a set of node-disjoint paths to choose from in the case of a failure on the primary path. The route discovery process is initiated by the BS that broadcasts a beacon message. Upon hearing the beacon message each of the first hop neighbors of the BS includes its ID in the beacon (in a newly added field that was left empty by the BS) to distinguish the branches of the tree rooted at the BS. The parent of a node that receives the beacon message is set to be the node from which it received the beacon. A node learns an alternative node-disjoint path to the BS if it receives a beacon from the same route update round but with a different first-hop ID. This way all nodes that can hear beacon messages from multiple branches know multiple node-disjoint paths to the BS. To enhance the chances of other nodes that cannot receive beacons from more than one branch, every node that discovers an alternate path broadcasts this information. Upon hearing an alternate route update message, a node checks if the message was received from a node different from its parent (to guarantee node-disjointedness); if so, the new route is added to its routing table, and its next hop on the alternate path is set to be the node from which it received the route update message. After that the route update message is rebroadcast so that other nodes can benefit from it. Figure 2.4 shows a simple network, in which the sink has 2

neighbors, i.e., 2 branches are constructed. The solid links represent a branch, the dotted links represent another branch, and the dashed links represent the available links between nodes. In the figure, Black nodes are the nodes that are able to hear beacons from 2 branches, and thus learn an alternate path directly. Grey nodes learn alternate paths from alternate route update messages. If a node wants to forward a data packet and its parent has failed, Per-hop Alternate Path Packet Salvaging (PAPPS) is done, the node randomly chooses an alternate path from its routing table, such that it does not have any node in common with the nodes on the route from the source to the current node, and thus, avoids cycles. Assume that link (A, C) has failed in Figure 2.4, then node A uses its alternate path through node B.

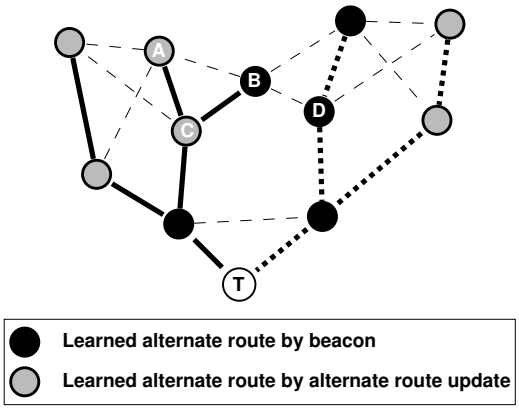


Figure 2.4 As long as the remaining credit ratio is larger than the remaining distance ratio, the forwarding mesh width increases. When it is less, each node uses its least cost path to the sink, and these paths might start merging as they approach the sink

Reactive protection can be applied in a different context other than recovering from path failures. Lost association with service nodes, such as APs or cluster heads, can be recovered quickly if a network node knows other service nodes within its range in advance. An example on such a scheme is found in (35). In this paper a fault-tolerant clustering mechanism for WSN was proposed. In a WSN each sensor node is associated with a *Gateway (Cluster Head)*. To preserve the scarce sensor energy, a sensor associates itself with the cluster head that can be reached using minimum transmission energy. The sensors associated with a gateway are said to be in the final set *FSet* of that gateway. The sensors that can be reached by a certain gateway,

say G_i but use less energy to reach another gateway, say G_j, are said to be in the backup set *BSet* of G_i, where the union of Bset and Fset is the range set *RSet*. Upon a gateway node failure or a range failure (the sensor node cannot reach its initial gateway) the sensor associate itself with the gateway that needs minimum energy to be reached (i.e., the sensor needs to be in at least one BSet). Otherwise, the failure cannot be recovered from.

2.3 Restoration or Recovery Mechanisms

Restoration mechanisms are implicitly implemented in all routing algorithms as route maintenance procedures. Ad hoc On-Demand Distance Vector (AODV) Routing uses either local or end-to-end restoration depending on the design parameter "MAX_REPAIR_TTL". This parameter represents the radius (in hops) around the destination in which intermediate nodes are allowed to do local recovery if a failure was detected on the used route. This parameter in turn depends on the network diameter. After local recovery is done and a new path is obtained, the length of the new path is compared to the length of the old path. If the new path has a higher number of hops, a RERR message is sent to the originating source to inform it about this change. Upon receiving the RERR, the source can choose to either keep the new route, or can initiate a new path discovery process. If the new path has similar number of hops as that in the old path, the recovery process will be invisible to the originating source and it will not be notified. The route maintenance procedure in dynamic source routing (DSR) is a combination of restoration and reactive protection and will be discussed in the following section.

As in protection mechanisms, restoration can be done to recover association with service nodes. In (36), the authors presented a scheme to provide survivability against AP failures. They presented *SAWAN* (Survivable Architecture for Wireless LANs). Basically, upon network deployment and before any failure, the AP should identify two kinds of nodes, 1) Bridge nodes, which are nodes that can hear from more than one AP, and 2) Leader nodes, which will act as control heads after the AP fails, and are responsible for calculating new routes to the remaining network. The authors suggested that, the associated nodes to a certain AP should switch to Ad hoc mode upon detecting the failure of that AP, and try to connect to the remaining network

with the aid of the leader and bridge nodes.

2.4 Hybrid Mechanisms

A mix of protection and restoration techniques can be used in this case. The main advantage of hybrid mechanisms is the design flexibility they provide, which helps in tailoring survivability mechanisms to fit certain application needs. In dynamic source routing (DSR), when a failure is detected by an intermediate node, this node sends an RERR message to the source. In addition, if the node that detected the failure has an alternate path to the destination in its memory, it uses this path to salvage the packet that triggered the RERR. However, if no routes are available then the packet is discarded. When the source receives the RERR it initiates a new route discovery process to restore its connectivity to the destination. A modification to this operation was proposed in (38). Upon detecting a failure a node attempts to repair (salvage) the failed route using information in its cache. If no route was found, bypass routing (restoration) is done without sending an RERR to the originating source. A prototype called SLR (Source routing with Local Recovery) is proposed, which is essentially a variation of DSR. This differs from DSR in being a little bit more optimistic, since no RERR message is sent to the source if salvaging fails. Simulation results show that this algorithm reduces the number of broadcasts done for path maintenance, and thus the number of route requests. In addition it was shown that it has a higher delivery ratio and goodput compared to DSR.

This combination of reactive protection, packet salvaging and restoration was also proposed in the CHAMP (Caching And Multi-Path) routing protocol (39). Basically, the protocol exploits temporal locality to help in salvaging a packet with a failed route (dropped packet), which is done through caching a number of recently forwarded packets at each node. When a failure occurs on the used route the affected node tries to salvage the packet using routing information in its memory. If it fails, it sends back an RERR message to the originating source, which contains the header information of the affected packet(s) that used that failed route. Upon receiving an RERR message, an upstream node checks to see if it has this packet in its cache. If so, it checks to see if it has an alternative route to the destination in its route cache

and sends the packet on this route. Otherwise, the RERR message is sent back to the next upstream node until it reaches the source. During route discovery a node that rebroadcasts a RREQ message keeps track of the minimum forwarding count ($fc_{\min}$) and the node(s) that sent an RREQ message with $fc_{\min}$ in a set P. This is done per destination. The set P will be used to distinguish the nodes that should forward the RREP back to the source. The same thing is done when dealing with the RREP messages from destination, i.e., the hop count hc to the destination is monitored, thus creating multiple source-destination paths of the same minimum length. Each node on the route keeps track of its next possible hops in a set S, and it alternates between them in a round robbin manner to forward packets to the same destination. This helps in distributing power consumption and the burden of extra storage at the nodes.

A dynamic policy-based multi-layer self-healing mechanism was proposed in (37). It was suggest that recovery from a failure can be done in different layers according to the survivability needs for the affected application (or applications). The mechanism is multi-layer because it uses different survivability schemes in layers 1, 3 and 4, where it was recommended that SCTP (10) (Stream Control Transport Protocol) should be used instead of TCP in layer 4. The authors suggested choosing from 1:N protection in L1, dynamic on demand re-routing in L3 or the multi-homing ability of SCTP in layer 4, depending on the nature of the running application. For example if the application is delay sensitive, 1:N should be chosen, while if it is delay tolerant the multi-homing ability of SCTP would be more suitable.

2.5 Survivable Backbones

Another problem that was studied in the literature is the problem of constructing a reliable network backbone. For example, in (42) the authors presented centralized and distributed algorithms to compute k-vertex connected spanning subgraphs. Simply, a k-vertex connected subgraph is a generalization of the minimum spanning tree, which is 1-connected. A different approach to solve the reliable backbone problem, is by finding a k-connected dominating set. This problem was further extended in (43), where the authors presented two algorithms to construct a k-connected m-dominating set $kmCDS$ in a graph G(V,E) to act as a communication

backbone for a WSN. A set $D \subset V$ is an m-dominating set if any node in $V \backslash D$ is a neighbor to at least m-1 nodes in D (a node dominates itself). The centralized algorithm CGA constructs a $kmCDS$ in $O(|V|^{3.5}|E|)$ by adding nodes to the set C (which will be the kmCDS when the algorithm ends) in a non-increasing order of their number of neighbors; breaking ties by the remaining power, and finally breaking ties arbitrarily by the node ID. The finishing step is to optimize C by removing nodes from it such that it remains k-connected and m-dominating. The authors propose a Distributed Deterministic Algorithm DDA to do the same job by first using one of the known distributed algorithms to compute a CDS, then using another known distributed algorithm to compute m-1 MISs (Maximum Independent Sets) in $G \backslash C$, and finally adding nodes to C relying on the fact that if a node has k neighbors in C then it can be added to it and the new C will still be k-connected. The difference between these algorithms and previous ones in the literature is that they allow the case of $k \neq m$.

2.6 Coding-based

The work in (22) gives an example of using erasure code,s in a fashion similar to that discussed in Section 1.3 in order to reduce the overhead of data redundancy. That is, a data packet is first divided into n smaller sub-packets, from which m redundant sub-packets are computed. Then, these $n + m$ sub-packets are sent on $n + m$ node disjoint paths, which will result in less overhead compared to duplication, especially if $m << n$. This enables us to tolerate m failures, since the destination needs only n sub-packets to recover the original data. In (22), the value of n is made equal to the expected number of paths that will be successful with high probability, which can be estimated given a set of paths to the sink and their failure probabilities. In (11) the authors introduced a network coding scheme that can be applied to both unicast and multicast connections to enhance their survivability against packet loss. The scheme allows a node to randomly generate linear combinations from the available packets in its memory, whenever it has an opportunity to transmit. The proposed approach can be applied to wire line and wireless packet networks since it considers both point-to-point and broadcast networks.

Table 2.1 Survivability mechanisms. In the table, PP=Proactive Protection, RP=Reactive Protection, Rt=Restoration, Hy=Hybrid, U=Unicast, M=Multicast, B=Broadcast, C=Convergecast, A=Association, Bk=Backbone, L=Link failures, N=Node failures and S=Service node failures

Scheme	Network Type	Class	Connection Type	Failure Type
Minimum-Energy Disjoint paths (31)	Ad-hoc	PP	U	N/L
AOMDV (25)	Ad-hoc	PP	U	L
AODVM (24)	Ad-hoc	PP	U	N/L
MP-DSR (40)	Ad-hoc	PP	U	N/L
GRAB (16)	WSN	PP	U	N/L
(17)	WSN	PP	U	N/L
Angle-based (41)	WSN	PP	U	N/L
Redundant Trees (32)	Ad-hoc	PP	M/B	N/L
FLSS (42)	Any	PP	Bk	N/L
KMDS (43)	WSN	PP	Bk	N/L
CTMP (33)	WSN	PP	C	N/L/S
EC (22)	WSN	PP	U	N/L
NC (11)	Any	PP	U/M	N/L
SMR (34)	Ad-hoc	RP	U	N/L
PAPPS (18)	WSN	RP	C	N/L
Fault-tolerant clustering (35)	WSN	RP	A	S
SWAN (36)	WLANs	Rt	A	S
AODV (13)	Ad-hoc	Rt	U	N/L
DSR (12)	Ad-hoc	Hy	U	N/L
PBMLSH (37)	Any	Hy	U	N/L
SLR (38)	Ad-hoc	Hy	U	N/L
CHAMP (39)	Ad-hoc	Hy	U	N/L

2.7 Summary

In this chapter, we covered the different classes of survivability mechanisms for multihop wireless networks, namely, proactive and reactive protection, restoration, hybrid and coding-based mechanisms. Selected examples from the literature were surveyed, which covers most of the survivability mechanisms spectrum. Table 2.1 summarizes all of these protocols and algorithms.

CHAPTER 3. Network coding-based protection of many-to-one flows

As discussed in previous chapters, traditional protection schemes are either resource-hungry like the (1+1) protection scheme, or introduce latency and interrupt the network operation like the $(1 : N)$ protection scheme. In this chapter, we present a network coding-based protection technique that overcomes the deficiencies of the traditional schemes. We derive and prove the necessary and sufficient conditions for our solution on a restricted network topology. Then we relax these connectivity requirements and show how to generalize the sufficient and necessary conditions to work with any other topology. We also show how to perform deterministic coding with {0,1} coefficients to achieve linear independence using a coding tree algorithm. Moreover, we discuss some of the practical considerations related to our approach. Specifically, we show how to adapt our solution when the network has a limited min-cut; we therefore define a more general problem that takes this constraint into account, which prove to be NP-complete. Furthermore, we discuss the decoding process at the sink, and show how to make use of our solution in the upstream communication (from sink to sources). Finally, we also study the effect of the proposed scheme on network performance.

3.1 Introduction

The many-to-one communication mode is used in a number of networks including two of the newer types of networks. The first is Wireless Mesh Networks (WMNs) (4), which are usually deployed to provide last-mile service to end users. WMNs are composed of Wireless Mesh Routers that form an infrastructure, which in turn is used to serve the Wireless Mesh Clients. In a WMN a router is called a gateway if it is connected to the wired network, where gateways provide Internet access to other routers through wireless multihop communication.

Note that the traffic in a WMN is either many-to-one from the wireless mesh clients to the gateway, or one-to-many from the gateway to the wireless mesh clients.

The second is Wireless Sensor Networks (WSNs) (15), which are composed of a large number of sensing nodes that are deployed in a specific area of interest to monitor a certain phenomenon or to report the occurrence of certain events. Sensors in WSN can work in two modes; 1) they can continuously (e.g., periodically) acquire and send information from their surrounding environment to the base station (BS), or 2) they can be event-driven, where only upon detecting an event (e.g., an intruder) the sensing nodes in the surrounding region send the acquired data to the BS. In the continuous mode all the sensors send data to the BS, while in the event-driven mode only the sensors in the vicinity of the event send data to the BS.

Wireless mesh clients or sensor nodes are information sources that need to send their data to the destination (the gateway in a WMN or the BS in a WSN). However, the wireless communication medium is prone to various types of interference causing a wireless-link status to dynamically change according to the channel conditions, hence resulting in information loss. ARQ (Automatic Repeat reQuest) and FEC (Forward Error Correction) may help in such scenarios, but they will not be of any benefit if the channel was down permanently (or for a considerable amount of time). If the channel's Bit Error Rate (BER) is high (due to interference), there will be too many errors in the received packet for the FEC to correct, and retransmission (by ARQ) will be only wasting the energy of the transmitting node.

In this chapter, we utilize network coding (26)(23)(47) to provide protection in a proactive manner to many-to-one flows. The main advantage of using network coding is in reducing the needed resources to provide such protection. This is a new application of network coding in wireless networks, and to the best of our knowledge, using network coding in this direction has not been explored. The main result in this chapter (Theorem 1) is summarized as follows:
In a many-to-one flow network with n sources, the single destination (sink) will be able to recover the n data units (from the n sources) in the case of a single link failure, if and only if, any subset of the n sources of size k can reach the sink through a set of edge-disjoint paths of size at least $k+1$, for all values of k such that $1 \leq k \leq n$.

The rest of the chapter is organized as follows. In Section, we describe the problem under consideration. In Section 3.3, we discuss the sufficient and necessary conditions for our solution to exist. Three generalizations of the original problem are discussed in Section 3.4. In Section 3.5 we show how to perform network coding using $\{0, 1\}$ coefficients. Section 3.6 discusses some of the practical issues related to our approach. In Section 3.7, we study effects of our protection mechanism on the network performance. Finally, we summarize the chapter in Section 3.8.

3.2 Problem Description

Link failures may occur due to severe channel fading, high levels of interference caused by other devices using the ISM band, or even physical damage to the network nodes or their antennas caused by harsh weather conditions. These problems may last for a considerable amount of time and they cannot be relieved using FEC or ARQ. Our objective is to efficiently provide protection against such link failures in a proactive manner using the minimum number of paths. We accomplish this by using network coding.

Let us consider the following motivating example shown in Figures 3.2(a), 3.2(b), 3.2(c) and 3.2(d). In this network there are two sources, S_1 and S_2 that need to send two data units, b_1 and b_2, respectively to a sink node T. To provide proactive protection against a single link failure each source must use the network in a different time slot to have two edge-disjoint paths to the sink as shown in Fig. 3.2(b). This is because the minimum-cut between the sources and the sink is 3. However, unlike proactive protection, if we want to provide reactive protection the two sources can use the network in the same time slot as shown in Fig. 3.2(c). However, if a failure takes place on one of the primary paths the affected source will have to detect the failure first, and then reroute its data to use the backup path through node A, which introduces delay and interrupts network operation. Now suppose that we allow node A to combine b_1 and b_2 (bitwise XOR), and send the resulting symbol to the sink on the link (A, T), as illustrated in Fig. 3.2(d). This way, the two sources can use the network in the same time slot and still achieve proactive protection. If any of the three symbols sent to the sink is lost due to a link failure, the sink will still be able to recover the original data units. For example, assume that

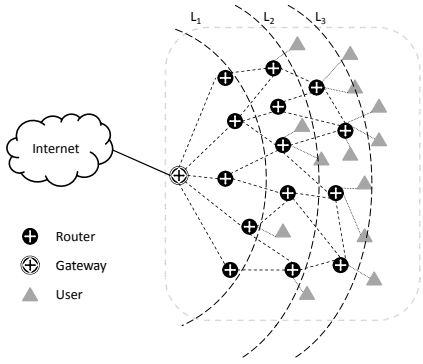


Figure 3.1 Wireless Mesh Network

link (S_2, C) fails, the sink will receive $b_1 \oplus b_2$ on link (A, T) and b_1 on link (D, T), and it can recover b_2 by performing the bitwise XOR operation on the received symbols.

Although the analysis in the following sections is applicable to both WMNs and WSNs (and any other network that supports the many-to-one flow structure), we focus our discussion only on WMNs. In addition, we focus our discussion on link failures although the theorems and lemmas can be easily extended for the node failure case. Specifically, we consider a WMN, in which there is only one gateway, and in which the routers can be organized in t levels, where the routers in level i are i hops away from the gateway (e.g., the network in Fig. 3.1 has 3 levels). In addition, we assume that each level of routers has a set of associated users, which communicate with each other through the routers only. From now on, we refer to the routers and users in level i by L_i and U_i respectively. We assume that the router nodes work on two frequency channels, one for the communication between the routers themselves to construct and use the underlying infrastructure, and the other to communicate with users so that users do not interfere with routers. Moreover, we assume that the t levels access the wireless medium in a TDMA manner, where each level of routers is assigned a different time slot, that is used to send data units from those routers, i.e. we study one level at a time. Actually, we assume that in each time slot the users transmit first (according to a schedule that will be discussed in Chapter 4), and routers can start their transmissions afterwards.

In general, since each level of routers and their associated users are active alone in their assigned time slot, we assume that there are n source nodes in the network, which represent

the users in the active level. We assume that each user generates a single data unit. Therefore, there are n data units from the n users that should be forwarded to the gateway router. Fig. 3.2(a) shows a network with two source nodes.

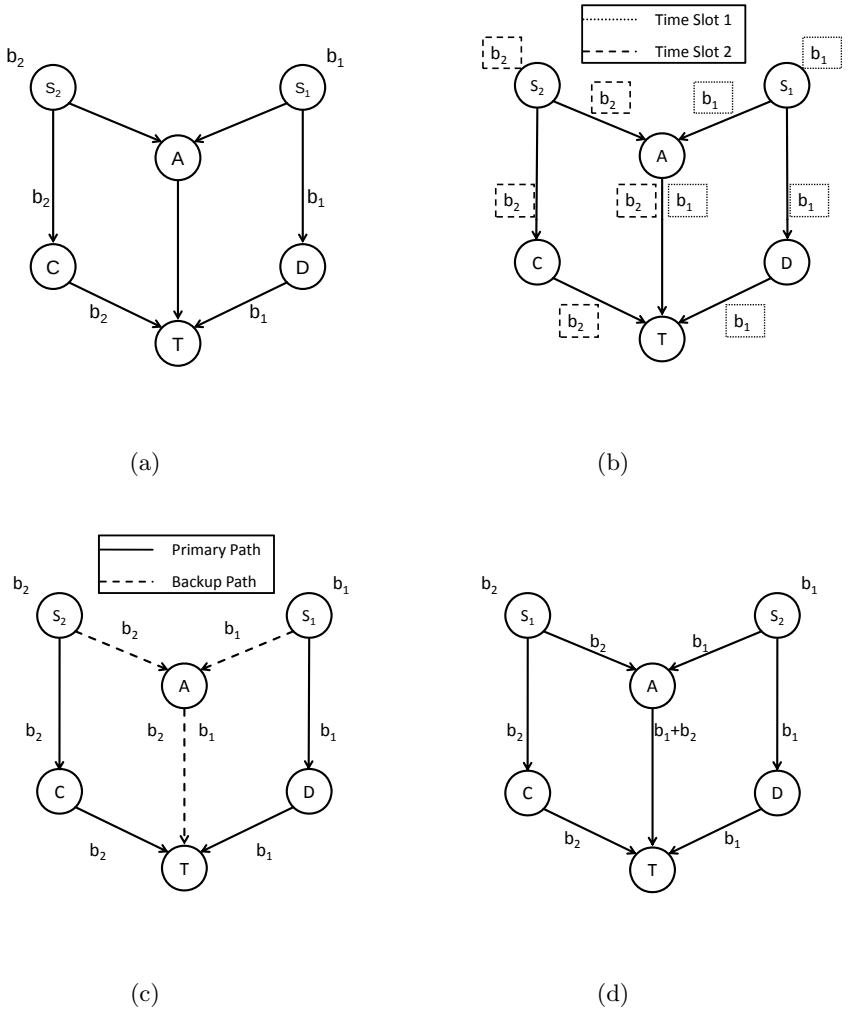


Figure 3.2 (a) Original graph (b) 1+1 Protection, (c) 1:2 Protection, (d) Network coding Protection

3.3 Proposed approach

In this section we start by developing our solution on a restricted network topology, which assumes the satisfaction of some connectivity and topology requirements (as will be stated below). Then we show how to relax each of these requirements, and provide an appropriate

generalization in Section 3.4.

3.3.1 Assumptions, Definitions and Notation

Since we are interested in the many-to-one flow from the users in U_i to the gateway, we can adopt the directed graph model in which a graph $G(V, E)$ is used to represent the network. The set of vertices V represents the network nodes (users and routers), and the set of edges E represents the available wireless links between network nodes, such that the edges are always directed from levels with higher indices to levels with lower indices and from users to routers in a certain level. Taking that into account we define the following:

1. Let U_s be the set of users in the level being considered, where $|U_s| = n$.

2. Let T be the only sink node in the network.

3. Let L_s be the set of routers in level s, where $|L_s| \geq n + 1$. In practice, this assumption is not always true and the reason to make such an assumption will become clear shortly. We relax this assumption in Section 3.6.

4. All the links in the original graph G are of unit capacity, and there are no parallel links.

5. The minimum link cut (or the minimum node cut, if we are concerned with node failures) between the nodes in L_s and the sink node T is $\geq n+1$. Networks that do not have this property are discussed in Section 3.6.

6. The sub-graph induced by the nodes in U_s and L_s is bipartite (general cases will be discussed later in this chapter). In reality, this assumption is half true since users in a WMN communicate only through routers, i.e., in terms of edges on graphs, no two user nodes have an edge in between. However, routers in the same level may communicate with each other, and the graph therefore may have edges between router nodes; we call such a graph a semi-bipartite graph. We show in the appendix a simple procedure to find an equivalent bipartite graph for any semi-bipartite graph that has edges between router nodes.

7. Only one link fails at a time.

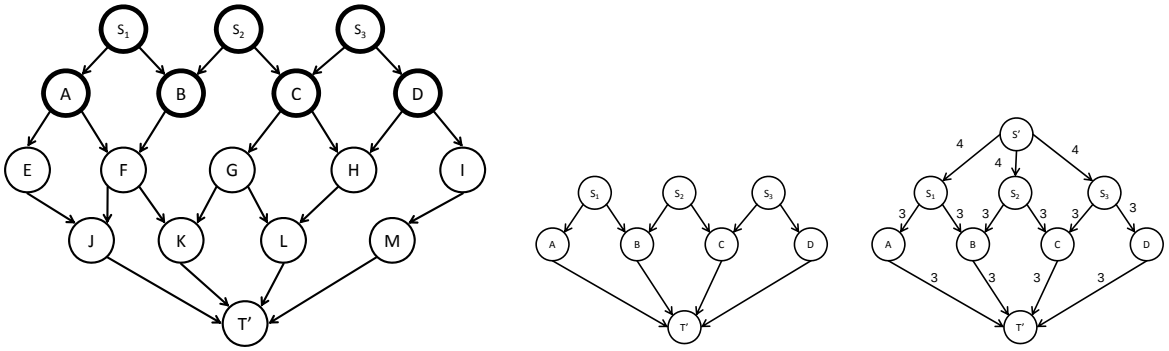
8. A node (either a user or a router) can receive from, or transmit to, multiple nodes simultaneously. This can be done by using multiple transceivers at each node. In Section 4 we show how to handle nodes with single transceivers.

9. All packets have the same length.

10. G^T is the graph formed by: the nodes in U_s and L_s and all the links between them, a hypothetical sink node T' and hypothetical links from all the nodes in L_s to T'.

11. G^{ST} is the graph formed by: the nodes in U_s and L_s, and all the links between them, with a capacity of n assigned to each of these links, a hypothetical sink node T', hypothetical links with capacity of n from all nodes in L_s to T', a hypothetical source node S' and hypothetical links with capacity of n+1 from S' to the nodes in U_s.

As an illustration of points 10 and 11 above, a simple graph is shown in Fig. 3.3(a), and its corresponding G^T and G^{ST} are given in Fig. 3.3(b) and Fig. 3.3(c) respectively. In these figures S_1, S_2 and S_3 are the nodes in U_s, and A, B, C and D are the nodes in L_s.



(a) $G(V, E)$ (b) G_{01}^T (c) G_{01}^{ST}

Figure 3.3 (a) The original graph G, (b) Graph G^T and (c) Graph G^{ST}

3.3.2 Sufficient and necessary conditions

Suppose we can deliver to the sink $n+1$ linear combinations (or equations) of the original n data units on $n+1$ edge-disjoint paths, such that, any n combinations are linearly independent (solvable). The sink can recover the original n data units by solving any n from the $n+1$ linear combinations. Since the $n+1$ paths are edge-disjoint, a single-link failure will affect at most one path. That is, the sink will still receive n linearly independent combinations and will be able to recover the original n data units. As in the $1+1$ protection scheme the recovery can be done without the need to detect the failure, and compared to the $1:N$ protection scheme, this approach requires the same number of paths $(n+1)$, but does not impose a delay or interrupt the network operation. This clarifies the basic idea of our approach. An example is shown in Fig. 3.2(d).

We divide the problem into two sub-problems. The first deals with the needed information content in the linear combinations, i.e., how should the data units be incorporated in the combinations to guarantee the successful recovery of the original n data units in the case of a failure. The second is the coding problem to guarantee the linear independence of any n combinations from the $n+1$ linear combinations. In this section we focus on the former and leave the latter to Section 3.5. Thus, we always assume that the created combinations are linearly independent in this section.

As mentioned earlier, only one level of users, U_s, is active at a certain time, and our goal is to use deterministic network coding to provide proactive protection for the n users in that level. Under assumption 6, coding cannot begin in sources, since each of which only knows its own data unit and does not have any knowledge about the other data units in other sources. Therefore, creating the $n+1$ combinations is the responsibility of the intermediate network nodes that connect the sources to the sink.

It is better to do the coding as close as possible to the sources, since this will reduce the used network resources as we will show in Section 3.6. Thus, we consider the closest nodes in the intermediate network to U_s that can perform coding on the data units, i.e., the nodes of L_s. We assume the case when $|L_s| = n+1$, i.e., each one of the nodes in L_s is responsible

for producing one combination and forwarding it to the sink. From assumption 5, each of the nodes in L_s has its own path to the sink that is edge-disjoint from the paths used by other nodes. Therefore, for simplicity, our original graph G can be replaced with G^T, where a path from a node in L_s to the sink is represented by a direct link. Taking this into account, the condition that will enable the L_s nodes to construct the $n+1$ combinations that can tolerate a single link failure is:

Condition: *Any k nodes in U_s, must be connected to at least $k+1$ nodes in L_s:* Consider the network in Fig. 3.4, if either of the links AT or BT fails, the sink will not be able to recover all three data units, because the other link that did not fail will be carrying the only combination of the two data units b_1 and b_2, while the sink needs at least two. Consider the linear combination created in a node v in L_s. There are n possible participants that can contribute to creating this linear combination. Let us assume that the combination consisted of two data units (i.e. v is connected to two source nodes in U_s). Then the sink can recover the two data units upon the failure of the path from node v if these two symbols were present in at least two other equations such that there are n independent equations in n unknowns. That is, if node v is connected to k nodes in U_s, then for the sink to be able to recover all the original data units if the combination created in v is lost, the neighboring set in L_s which encode data units from the k nodes in U_s must be of size at least k, or $k+1$ if we include v. In general we can say: *Any group of nodes in U_s of size k must be connected to at least $k+1$ nodes in L_s.*

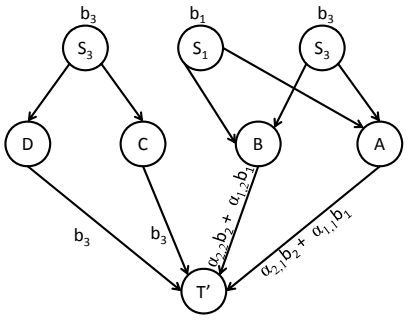


Figure 3.4 Condition not satisfied

Using the concept of matching in graph theory, an equivalent statement would be: $n+1$ *perfect matchings between the nodes in* U_s *and those in* L_s *must exist, such that each matching,* M_i*, corresponds to the case when one of the nodes in* L_s*,* v_i*, is removed, where* $1 \leq i \leq n+1$. This guarantees the existence of a dedicated alternate path from every source node to the sink upon a single link failure. This condition implies that the Max-flow is greater than or equal to 2 from every source to the sink.

We now continue with proving that this condition is necessary and sufficient for the nodes in L_s to be able to construct the $n+1$ combinations that can tolerate a single link (or path) failure.

Lemma 1. *The sink will recover the* n *data units even if one of the* $n+1$ *combinations is lost, if and only if, any subset of nodes in* U_s *of size* k *is connected to a subset in* L_s *of size at least* $k+1$*, for all values of* k*, where* $1 \leq k \leq n$.

Proof. In the previous scenario we view the data units from sources as variables, and the $n+1$ nodes in L_s as combinations (or equations), and a variable is present in an equation if the corresponding source is connected to the node representing that equation.

We prove the implication by contradiction. Assume that the sink is able to recover the n data units, even if one of the $n+1$ combinations is lost. But let there be a subset of U_s nodes of size k, that is connected to a subset of L_s nodes of the same size k. Then, the sink cannot randomly choose n combinations from the $n+1$, because it **MUST** pick all the k combinations that were formed by the subset of L_s nodes mentioned above; otherwise, the k variables from the corresponding k nodes in U_s will only be present in $k-1$ equations, i.e., they cannot be recovered. This contradicts the assumption that the sink is able to recover the original n data units if **ANY** of the combinations was lost, which concludes the proof of the implication.

To prove the converse, we also use contradiction. Assume that any subset of nodes in U_s of size k is connected to another subset of nodes in L_s that is of size at least $k+1$. But, there is a mandatory combination, which cannot be lost for the sink to be able to recover the original n data units. A combination is essential and can not be lost, if it leaves a set of equations of size say l with $l+1$ unknowns, which are impossible to solve without that combination. But

for this case to happen, there must have been some $l+1$ nodes in U_s that are only connected to $l+1$ nodes in L_s, which contradicts our original assumption, of having any k nodes in U_s connected to at least $k+1$ nodes in L_s, for all values of k, where $1 \leq k \leq n$. □

It can be seen that a naive check for the above condition takes time of order $O(2^n)$ since we must consider all values of k. We will now show how to check the condition above in polynomial-time with respect to the number of sources using a max-flow algorithm. The graph G^{ST} is used in the next Lemma. (see Section 3.3.1, bullet 11 for definition).

Lemma 2. *An S-T maximum-flow of at least $n(n+1)$ is achievable in G^{ST}, if and only if, any subset of U_s of size k is connected to a subset in L_s of size at least $k+1$, for all values of k, where $1 \leq k \leq n$.*

Proof. We prove the implication by contradiction. Assume the max-flow value is indeed $n(n+1)$, then all the links from S to the n original sources are saturated (i.e., each one carries a flow equal to $n+1$). And assume that there are some k nodes in U_s that are **only** connected to some k other nodes in L_s. The incoming flow to this component equals $k(n+1) = kn + k$ while the outgoing capacity equals kn, which means that there are k units of flow that will be blocked from the sink, that is the max-flow $= n(n+1) - k$ which contradicts the original assumption of the max-flow.

We prove the converse by contradiction. Suppose that any k nodes in U_s are connected to at least $k+1$ nodes in L_s, but the maximum achievable flow was less than $n(n+1)$, then there are some of the links from S to the nodes in U_s that could not be saturated. Assume only one of those links carried n units of flow to a node in U_s say node u. Then node u either has a single outgoing link, or is one of k nodes in U_s, that are connected to another set of k nodes in L_s (otherwise, it would have been able to forward this remaining unit of flow to the sink through an augmenting path on the residual network (48)). In both cases node u will violate the connectivity assumptions. Therefore, the max-flow must be $n(n+1)$. This concludes the proof. □

3.4 Generalizations

In this section we introduce three generalizations. First we discuss the case when $|L_s|$ is larger then $n+1$, and introduce a mixed integer linear program (MILP) to compute the minimum number of L_s that can be used to tolerate a single failure. Then we generalize the sufficient and necessary conditions presented in the previous section to be suitable for any network topology (not necessarily bipartite), and any number of failures.

3.4.1 The case of $|L_s| > n+1$

Until now, we have assumed that the number of nodes in L_s is exactly $n+1$. The number of L_s nodes could be larger than $n+1$. This however, does not invalidate our conditions and the above requirements will still apply.

The only difference is that the minimum number of combinations may be more than $n+1$, depending on the topology. One extreme case, is when each of the n sources is connected to exactly two nodes in L_s, and each of the L_s nodes has exactly one neighbor in U_s, i.e., $|L_s| = 2n$. Assuming that the max-flow from L_s to the sink is $2n$, the minimum number of combinations that tolerate a single link failure in this case is $2n$, which is equivalent to 1+1 protection, where coding is not needed and routing can be used.

Another issue that arises in this general case is selecting the appropriate linear combinations that will enable the sink to recover all the original data units. The network shown in Fig. 3.5 gives a good example, where the minimum number of combinations is $n+2$. In this network, selecting four combinations randomly may not cover all the data units. For example, if the combinations created in nodes C, D, E and F were chosen by the sink to calculate the original four data units, b_1 cannot be recovered. However, if any of the above mentioned four combinations was replaced by either of the combinations from A or B, the sink will be able to recover all the original data units. We conjecture that the problem of finding the minimum number of L_s nodes that satisfies the connectivity conditions, and hence, the problem of finding the minimum number of linear combinations that can tolerate a single link failure is NP-Complete. We therefore formulate a solution to this problem as a mixed integer linear

program.

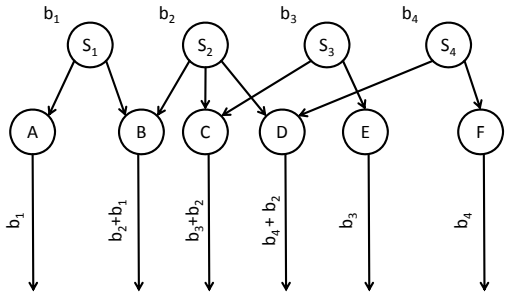


Figure 3.5 Minimum number of combinations = n+2

We use the graph G^{ST} to formulate an MILP to calculate the minimum number of paths that are needed from nodes in L_s to the sink, in order to forward $n(n+1)$ units of flow. Each node in L_s is traversed by a single path to the sink, and thus calculating the minimum number of paths will result in the minimum number of nodes in L_s that satisfies the connectivity conditions.

3.4.1.1 Notations

- Let m be the number of nodes in L_s.
- $N_a^b(u)$ is the set of neighbors in a of node u, such that u is in b, where b and $a \in \{U_s, L_s\}$ and if $b \in U_s$, then $a \in L_s$ and vice versa.
- f_{uv} and c_{uv} corresponds to the flow and capacity of edge (u, v) respectively, where $0 \leq f_{uv} \leq c_{uv}$.
- $c_{S'u} = n + 1, \ \forall u$ where S' is the hypothetical source and $u \in U_s$. The capacity of all other edges is n.
- y_v is defined for every node v in L_s and it equals the sum of all flows going into that node.
- z_v is a binary variable that is defined for every node v in L_s, which is equal to 1 if the outgoing link from v to the sink carries at least one unit of flow.

3.4.1.2 MILP

We begin with the assumption that the maximum achievable flow from S' to T' is $n(n+1)$. All the nodes in L_s that participate in forwarding the flow are selected to be the coding nodes. We calculate the minimum number of L_s nodes that satisfies the connectivity conditions by calculating the minimum number of used paths. The objective function is:

$$Minimize \quad \sum_{v=1}^{m} z_v \tag{3.1}$$

Subject to:

$$z_v - \frac{y_v}{n} \geq 0, \forall v \in L_s \tag{3.2}$$

This constraint is defined for every node v in L_s, and it sets the binary variable z_v to 1 if there is an outgoing flow from node v to the sink T', i.e., the path from v is used.

$$\sum_{\forall v: v \epsilon N_{L_s}^{U_s}(u)} f_{uv} = n+1, \forall u \in U_s \tag{3.3}$$

$$y_v - \sum_{\forall u: v \epsilon N_{L_s}^{U_s}(u)} f_{uv} = 0, \forall v \in L_s \tag{3.4}$$

These two constraints represent the conservation of flow constraints, where the constraint in equation 3 beside conserving the flow assures that the links from the hypothetical source to all the nodes in U_s are saturated to guarantee a max-flow of $n(n+1)$. And, the constraint in equation 4 (combined with the bounds on y_v below) restricts the sum of all incoming flows to a certain node v in L_s not to exceed the capacity of the single outgoing link to the sink. Finally the following two bounds are needed:

$$0 \leq f_{uv} \leq c_{uv}, \forall (u,v) \tag{3.5}$$

$$0 \leq y_v \leq n, \forall v \tag{3.6}$$

For $u \epsilon U_s$ and $v \in L_s$.

3.4.2 General Network Topology

Assumption 6 in Section 3.3.1 states that the graph induced by the nodes in U_s and L_s is bipartite. Although this topology is suitable for the assumed scenario of routers and user (in a WMN), it is a restricted topology that may not always apply to other many-to-one flows in different types of networks. Therefore, we need to generalize the conditions in Lemma 1 for other network topologies.

By carefully inspecting the condition of Lemma 1, one can see that the essence of the solution lies in the number of edge-disjoint paths (or node-disjoint paths if we are concerned with node-failures) from a group of sources to the sink. In the special case considered previously, each node in L_s represented one such path. Hence, Lemma 1 can be generalized in the following Theorem.

Theorem 1. *The sink will be able to recover the n data units even if **ANY** one from the $n+1$ combinations is lost, if and only if, any subset of nodes in U_s of size k is connected to the sink through a set of edge-disjoint paths of size at least $k+1$, for all values of k, where $1 \leq k \leq n$.*

Proof. The proof follows directly the same reasoning used in proving Lemma 1. □

As an example to illustrate Theorem 1, consider a less restricted network topology, where we allow links between sources, as shown in Fig. 3.6. Table 3.1 lists a collection of edge-disjoint paths between every possible combination of sources and the sink. It can be easily verified that the network in Fig. 3.6 satisfies the condition in Theorem 1. Moreover, this condition can be checked using the same idea in Lemma 2. Specifically, by assuming 1) that each source node is connected to a hypothetical source S' through a link with a capacity of $n+1$, and 2) that all other links have a capacity of n. Then, checking if the max-flow to the sink is at least $n(n+1)$.

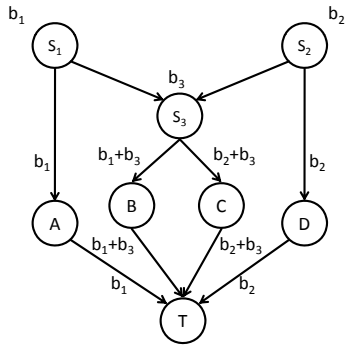


Figure 3.6 Non-bipartite topology: links are allowed between source nodes

Table 3.1 Edge-disjoint paths from combinations of sources to sink in Figure 3.6. Note that any k sources have k+1 edge-disjoint paths to T

Sources	Paths
S_1	$\{S_1$-A-$T\}$,$\{S_1$-S_3-B-$T\}$
S_2	$\{S_2$-D-$T\}$,$\{S_2$-S_3-C-$T\}$
S_3	$\{S_3$-B-$T\}$,$\{S_3$-C-$T\}$
$\{S_1, S_2\}$	$\{S_1$-A-$T\}$,$\{S_1$-S_3-B-$T\}$,$\{S_2$-D-$T\}$
$\{S_1, S_3\}$	$\{S_1$-A-$T\}$,$\{S_1$-S_3-B-$T\}$,$\{S_3$-C-$T\}$
$\{S_2, S_3\}$	$\{S_2$-D-$T\}$,$\{S_2$-S_3-C-$T\}$,$\{S_3$-B-$T\}$
$\{S_1, S_2, S_3\}$	$\{S_1$-A-$T\}$,$\{S_2$-D-$T\}$,$\{S_3$-B-$T\}$,$\{S_3$-C-$T\}$

3.4.3 Multiple Failures

The necessary and sufficient conditions for the case of multiple failures can be derived from our previous discussion, and are summarized in the following theorem:

Theorem 2. *The sink will be able to recover the n data units even if e link failures occur (i.e., at most e combinations are lost), if and only if, any subset of U_s of size k is connected to the sink through a set of edge-disjoint paths of size at least $k + e$, for all values of k, where $1 \leq k \leq n$.*

Proof. The proof follows directly the same reasoning used in proving Lemma 1. □

Although we have discussed three generalizations in the previous subsections, we continue our analysis of the baseline case that satisfies the assumptions in Section 3.3.1.

3.5 Coding

In the previous sections, we assumed the linear independence of any n combinations from the $n+1$ combinations produced at some L_i. In this section, we will show how to achieve this independence between combinations through using $\{0,1\}$ coefficients (binary network coding). Using binary network coding reduces all operations to bit-wise XOR operations, and simplifies the coding and decoding processes.

A linear combination is a summation of data units (b_i's) each of which is multiplied by a coefficient (α_i) from a finite field $GF(q)$, as follows: $C = \sum_i \alpha_i.b_i$, where $b_i, \alpha_i \in GF(q)$. The independence of combinations relies on the b_i's and the choice of the α_i's. Therefore, achieving independence using $\{0,1\}$ coefficients depends solely on how we compose each combination from only the data units, i.e., a data unit is present in a combination if its coefficient is 1, and a data unit is not present if its coefficient is 0. For instance, in Fig. 3.2(d), the three combinations that were sent to the sink are, $C_1 = b_1$, $C_2 = b_2$ and $C_3 = b_1 + b_2$, each of which is composed only from data units and no coefficients (other than 1 and 0) were used.

In the following subsections we assume that 1) the connectivity condition of Lemma 1 is satisfied, and 2) if the graph induced by the nodes in U_s and L_s is not bipartite, the transformation in the appendix is used to get the bipartite equivalent. We now show how to decide on the data units composing each of the linear combinations, through finding simple paths and trees.

3.5.1 The benefits of paths and trees

Consider a path in the bipartite graph that has both ends in L_s and in the middle it alternates between U_s and L_s until it includes all the nodes in U_s. It is clear that any k nodes from U_s on that path have at least $k+1$ neighboring nodes from L_s that are also on the same path. Also, note that the number of L_s nodes on such a path is the minimum number of nodes that satisfies the connectivity conditions, because each source has only two neighboring nodes in L_s.

Such a path not only finds a set of nodes in L_s that satisfy the connectivity conditions, but

also helps in the assignment of the coding coefficients to create the needed linearly independent combinations. To illustrate the benefits of coding according to the connectivity on a path, consider the example in Figure 3.7, where we have 4 source nodes in U_s, and 5 nodes in L_s. If we let all the sources use all their outgoing links to L_s as shown in the example, we will have dependent combinations like $\{b_4 + b_3\}$ and $\{b_4 + b_3\}$ (or $\{b_1 + b_2\}$ and $\{b_1 + b_2\}$), thus invalidating the linear independence requirements.

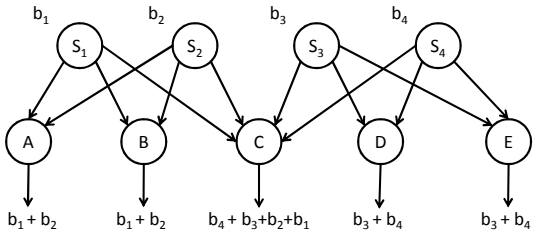


Figure 3.7 Bad coding: linear Independence of any n combinations is not satisfied

However, consider the simple path {A, S_1, B, S_2, C, S_3, D, S_4, E} that is represented by the solid edges in Figure 3.8. If we compose the combinations at the L_s nodes according to their connectivity with the nodes in U_s on the path, i.e., a solid line correspond to a coefficient of 1 and a dashed line correspond to a coefficient of 0, then linear independence will be guaranteed, since any two combinations cannot have more than one element (i.e., data unit) in common.

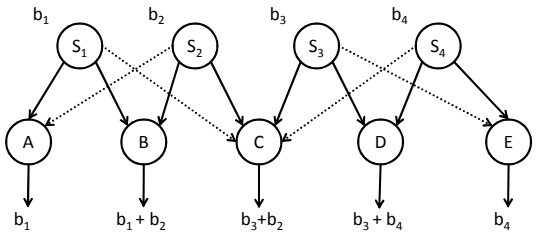
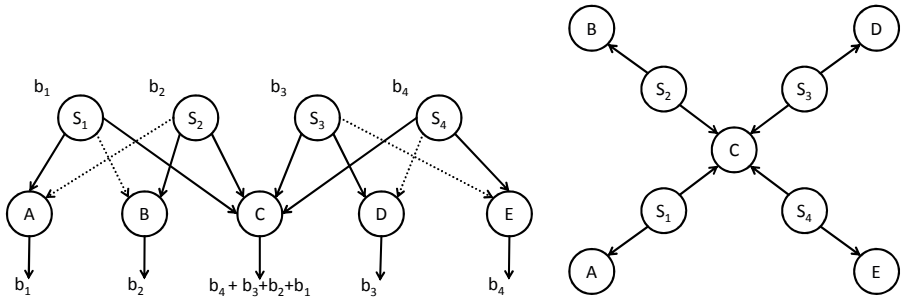


Figure 3.8 Path coding: linear independence is satisfied in any n combinations

Of course, we may not always find such a simple path. However, since a path is a special case of a tree with two leaves, then if we can find a tree that covers all the nodes in U_s, with all of its leaf nodes in L_s, we can construct independent linear combinations according to the

connectivity on the tree. This is shown in Figure 3.9(a), for the network presented in Figure 3.7, Figure 3.9(b) clarifies the underlying tree structure.



(a) (b)

Figure 3.9 (a) Tree coding: linear independence is satisfied in any n linear combinations, (b) Underlying tree

We now prove that constructing the linear combinations at the nodes in L_s according to their connectivity with the source nodes in U_s on the coding tree results in a set of $n+1$ linear combinations such that any n of them are linearly independent.

Theorem 3. *If the linear combinations at the nodes in L_s are created according to their connectivity with the source nodes in U_s on the coding tree, i.e., a link on the tree is assigned a coefficient of 1 and a link not on the tree is assigned a coefficient of 0, then any n combinations from the resulting $n+1$ linear combinations are linearly independent.*

Proof. A direct proof is used to prove this implication. We prove that any n combinations from the $n+1$ are linearly independent by proving that they are solvable by constructing an algorithm to solve for the n data units. In the algorithm the term *"leaf combination"* refers to a combination created at a leaf node in the tree, which will be a trivial combination that consists of a single data unit. The algorithm works as follows:

1. Put all the data units from the leaf combinations in a set; let us call it the *Recovered Data Units Set*, or the *RDU set* for short.

2. Remove all data units in the RDU set from the remaining combinations. This is done through XORing a data unit with all the combinations that it participates in.

3. After the previous step, a new set of data units will be recovered. We make these compose the new RDU set. The data units in the old RDU set will not be used further since they are removed from all combinations.

4. Repeat Steps 2 and 3 until all the data units are recovered.

To see how any n combinations are solvable, let us assume that one of the combinations created at the nodes in L_s is lost. There are two possibilities for this combination:

1. It is a leaf combination: in this case we are guaranteed to have at least one other leaf combination (when the tree is a path), and the decoding process can start from it.

2. It is a non-leaf combination: in this case the coding tree is divided into two smaller trees, each of which will have at least one leaf combination (when the tree is a path).

Note that the running time for this algorithm is $O(n)$, since in each step at least one data unit is recovered. The worst case occurs when the coding tree is a path and one of the leaf combinations is lost. □

3.5.2 Constructing a coding tree

We can construct a coding tree using the following three steps: first we begin by constructing a tree rooted at a node in L_s, then we modify its structure to guarantee that there are no leaves in U_s. Finally we trim the extra L_s leaves if any (when $|L_s| > n + 1$).

A tree can be constructed in time of order $O(|E|)$, e.g., a depth-first search (DFS) tree or a breadth-first search (BFS) tree, and the trimming can be done in time of order $O(|V|)$. The non-trivial part is modifying the structure of the tree to guarantee that there are no U_s leaves. Algorithms 1 and 2 describes a procedure to do the modification.

In Algorithm 1, we search the tree for a leaf node that falls in U_s. Upon finding such a leaf node u, we look for a neighbor x in L_s for node u that is different from the parent of u. We

Algorithm 1 Construct a coding tree

1: **while** there are leaves from U_s **do**
2: Pick a leaf node from U_s in the tree, say u
3: Find one of u's neighbors in L_s say x //other than u's parent
4: Call **ModTree**(u, x)
5: **end while**

Algorithm 2 ModTree(u,x)

1: Connect u to x, this will create a cycle, say C
2: Traverse the nodes on C, until we reach a node in U_s, say node v, that has a neighbor w not on the cycle.
3: **if** w is already connected to v **then**
4: Cut the cycle directly before or after v
5: **return**
6: **else**
7: Cut the cycle before or after v
8: Call **ModTree**(v, w)
9: **end if**

are guaranteed to find such a neighbor x, because each single node in U_s is connected to at least 2 nodes in L_s (Lemma 1). After finding u and x, the procedure **ModTree** adds the link between them to the tree creating a cycle C. Then, it traverses the nodes on C to find a node v in U_s that has a neighbor w in L_s not on C. Again, we are guaranteed to find such a node v that has such a neighbor w, because any k nodes in U_s are connected to at least $k+1$ nodes in L_s, and since the cycle is composed of equal numbers of nodes from both U_s and L_s, then there must be a U_s node on this cycle that has a neighbor in L_s not on the cycle. If v was connected to w on the tree, we cut the cycle before or after v, to make the graph a tree again. On the other hand, if v and w are not connected on the tree, we recursively call **ModTree**. As an illustration consider the network in Figure 3.10, the resulting DFS-Tree (Depth First Tree) rooted at node C is shown in Figure 3.11(a). The nodes that will be found when running Algorithm 1 and two iterations of **ModTree** are shown in Figure 3.11(b), the cycles and the edges that are marked to be cut are shown in Figure 3.11(c), and the final result after trimming the extra leaf nodes is shown in Figure 3.11(d).

There can be at most $n-1$ leaf nodes in U_s, and the recursive call to **ModTree** can be done at most n times. Hence, the running time of Algorithm 1 is of order $O(n^2)$. Note

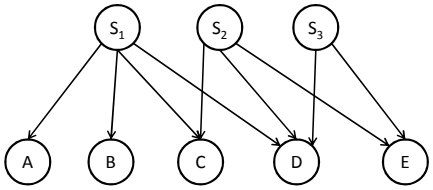


Figure 3.10 Network with three nodes in U_s, and five nodes in L_s

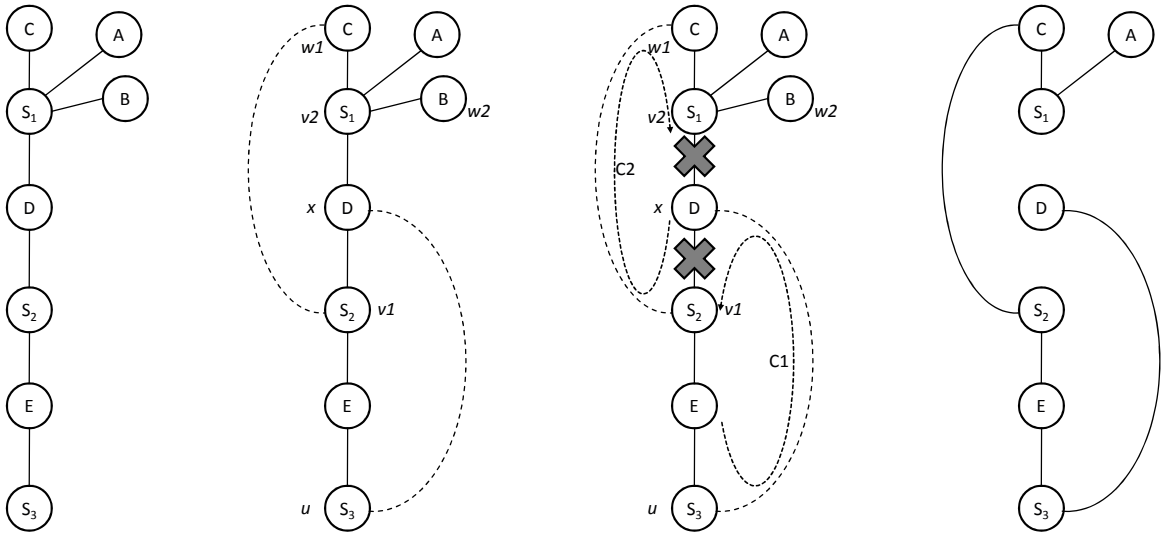


Figure 3.11 (a) DFS-Tree, (b) Two iterations: Note that links (S_3,D) and (S_2,C) are in the original graph but not in the tree, (c) Making the modification, (d) Result

that trimming extra leaf neighbors does not guarantee the minimum number of nodes in L_s. Therefore, to see how well this algorithm performs, in terms of needed number of L_s nodes, we compared it to the MILP presented in Section 3.4.1, the results are shown in Fig. 3.12. Each point on the graph corresponds to the average number of needed L_s nodes over 100 random topologies with the same number of nodes in U_s and L_s, where we varied the number of U_s nodes from 2 to 10 while keeping the number of L_s nodes twice as many.

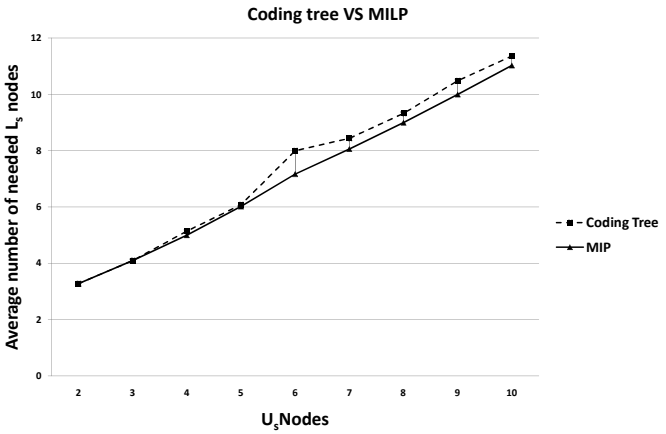


Figure 3.12 Comparing the number of needed L_s nodes in the coding tree to that computed by the MILP

3.6 Practical Considerations

In this section we start by considering networks with limited minimum cuts, where we show how to modify our solution to work even if the minimum cut between the sources and sink was less than $n+1$, and we formulate an MILP to solve this problem. In addition, we show that this problem is NP-complete by a simple reduction from the K-set cover problem. After that, we discuss the decoding process at the sink and we show that designing the combinations in a optimal manner that allows their fast recovery (according to the decoding process discussed in Section 3.5.1) is an NP-complete problem also. Finally, we show how to make use of our solution in upstream data transmission (from sink to sources).

3.6.1 Networks with limited minimum cuts

In our previous discussion, we always assumed that the min-cut between L_s and the sink is greater than or equal to $n+1$. However in practice this may not be the case, since in reality the network gets narrower as we approach the sink. Thus, in this section we study networks with limited min-cuts.

From Menger's theorem (48), the maximum number of edge-disjoint paths between the nodes in L_s and the sink is equal to the minimum edge cut. Let the number of these edge-

disjoint paths be h. If h is greater than or equal to $n+1$, then our approach can be applied directly, and the combinations formed in L_s can be forwarded to the sink. On the other hand, if h is less than $n+1$, then the formed combinations cannot be forwarded as is, and must be modified.

Let us assume that $h < n+1$, then the sink cannot receive more than h combinations at a time. That is, there is no point in allowing more than $h-1$ sources to transmit at the same time if we want to achieve protection using our scheme. Therefore, we propose to divide the n sources into groups of size $h-1$ sources each, and then choose a set of feasible groups that covers all the sources. We assume that the groups are time multiplexed, and we define a group of sources to be feasible if it satisfies the condition in Theorem 1. We say a set of groups covers all the sources, if each source is present in at least one of the groups in the chosen set.

The way we choose the covering set of feasible groups must take the following into consideration:

1. The degree of disjointedness between groups, which affects the fairness and the rate at which the sources transmit, as we will see in Section 3.7.

2. The used network resources.

As an illustration consider the network in Fig. 3.13, which contains four sources. The maximum number of edge-disjoint paths (h), from L_s to the sink in this network is equal to 3. Therefor, the largest possible group of sources, that can be protected together, will be of size at most 2. In this example all the groups of size two are feasible according to Theorem 1. Let us now compare the following three choices of sets:

1. Set_1=$\{\{S_1S_2\}, \{S_1S_3\}, \{S_1S_4\}\}$: In this set S_1 is common in all the groups, which is not a fair solution. This is because, if S_1 was allowed to transmit in all the three time slots, it will be transmitting at a rate of 1 *symbol/time slot*, while each of the remaining sources is allowed to transmit only in one of the three time slots, i.e, transmitting at a rate of $\frac{1}{3}$ *symbol/time slot*.

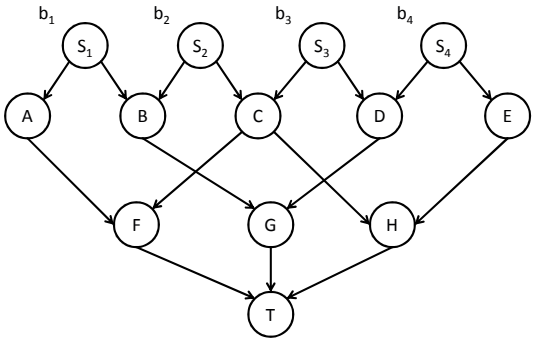


Figure 3.13 A network with four sources and $h = 3$

2. Set_2=$\{\{S_1S_3\}, \{S_2S_4\}\}$: this choice of groups achieves better fairness, where the bandwidth is equally divided and all the sources transmit at a rate of $\frac{1}{2}$ *symbol/time slot.*

3. Set_3=$\{\{S_1S_2\}, \{S_3S_4\}\}$: this is the best solution, because not only we achieves better fairness than Set_1, but also we use less network resources than Set_2, since each of the groups uses only three links to forward data to the minimum cut edges, compared to four links in the groups of Set_2.

The problem of choosing the smallest set of feasible groups to cover all sources can be proved to be NP-complete through a reduction from the **K-Set Cover** problem as will be shown in the next subsection.

3.6.2 Problem Complexity

In this section we show that the problem of dividing the sources into feasible groups, and choosing a covering set from the groups to cover all the source nodes is NP-complete. First, we start by presenting the decision version of the source grouping problem. We will call this problem the **Source-Grouping** problem, then we will consider a simplified version of **Source-Grouping** and prove that it is NP-complete by a reduction from the **K-Set Cover** problem.

Source-Grouping

Given: A graph $G(V, E)$, the set of source nodes U_s, the sink node T, and an integer h, which is equal to the number of edges whose removal disconnects the network, and leaves it divided into two partitions, one containing the sink node T, and the other containing the set of sources

U_s.

Question: Is there a collection of at most C groups, where each of the groups is of size $h-1$, such that:

1. All the groups are feasible, i.e. for each group any k sources in this group reaches the sink through a set of $k+1$ edge-disjoint paths, for $1 \leq k \leq h-1$.

2. The groups cover all the source nodes.

Theorem 4. *The **Source-Grouping** problem is NP-complete.*

Proof. This problem belongs to the $\mathcal{NP}$ class, since if we are given a collection of groups, we can check the covering condition in polynomial-time, by calculating the union of all groups and checking if it is equal to U_s. We can also check the feasibility condition for each group in polynomial-time by assuming that each source node has a supply of h units of flow, that needs to be forwarded to the sink node T on the graph edges, where each edge has a capacity equal to $h-1$ (from Lemma 2). This can be accomplished using a max-flow algorithm which has an $O(n^3)$ time complexity (if we use the pre-flow push algorithm (49)), and since there can be at most $n-h+2$ groups the total time will be of order $O(n^3.(n-h+2))$ which is dominated by $O(n^4)$.

To prove the NP-completeness of **Source-Grouping** it is enough to show that part of it is NP-complete. Hence, we will ignore the feasibility condition and we will assume that we are given the set of feasible groups, and our problem is confined to finding a collection of feasible groups that covers U_s. Thus, the new version of **Source-Grouping** which we will call **Source-Covering** can be defined as follows:

Given: The set of sources U_s, the collection of all the groups in U_s that are feasible, $F = \{G_1, G_2, \dots\}$, where $|G_i| = h-1, \forall i$, and a positive integer C.

Question: Can we choose at most C groups from F whose union gives L_s?

To prove that **Source-Covering** is NP-complete, we will show that any instance of the **K-Set Cover** problem can be mapped directly to an instance of **Source-Covering**. For completeness we state the **K-Set Cover** problem:

K-Set Cover

Given: A set of elements $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$, a collection of subsets of $\mathcal{E}$, $\mathcal{B} = \{B_1, B_2, \dots\}$, where $|B_i| = K$, $\forall i$, and a budget D.

Question: Can we find at most D subsets from $\mathcal{B}$ whose union gives $\mathcal{E}$?

Obviously, the set $\mathcal{E}$ in the **K-Set Cover** problems maps directly to the set U_s in **Source-Covering**. Also $\mathcal{B}$ maps to F, K maps to $h-1$ and D maps to C. Therefore, any instance of the **K-Set Cover** problem can be transformed into an instance of the **Source-Covering** in time of order $O(1)$, and finding a solution to any of them solves the other, which means that **Source-Covering**, and hence **Source-Grouping** are both NP-Complete.

□

3.6.3 MILP Formulation

In this subsection we formulate the problem of source grouping as a mixed integer linear program. For convenience, we define the following:

1. $s(k)$ source number k, where $1 \leq k \leq n$.
2. M the maximum number of groups which equals $n - h + 2$.
3. f_{ij}^{kc} the flow of source k in group c on edge (i, j).
4. z_{ij}^{c} a binary variable which is equal to 1 if the edge (i, j) carried flow for group c and 0 otherwise.
5. g_c^k a binary variable which is equal to 1 only if source k was in group c and 0 otherwise.

Assuming that the capacity of all edges is $h - 1$, the linear integer program is:

$$Minimize \sum_{c=1}^{M} \sum_{\forall (i,j) \in E} z_{ij}^{c} \tag{3.7}$$

Subject to:

$$\sum_{\forall j:(s(k),j)\in E} f_{s(k)j}^{kc} = g_c^k.h \ , \ \forall k, c \tag{3.8}$$

$$z_{ij}^c - \frac{\sum_{k=1}^{n} f_{ij}^{kc}}{h-1} \geq 0 \ , \ \forall c, (i,j) \in E. \tag{3.9}$$

$$\sum_{\forall i:(i,j)\in E} \sum_{k=1}^{n} f_{ij}^{kc} = \sum_{\forall i:(j,i)\in E} \sum_{k=1}^{n} f_{ji}^{kc} \tag{3.10}$$

$$\forall c, j \neq x \in \{T, s(1), \ldots, s(n)\}$$

$$0 \leq \sum_{k=1}^{n} f_{ij}^{kc} \leq h-1 \ , \ \forall c, (i,j) \in E \tag{3.11}$$

$$\sum_{c=1}^{M} g_c^k = 1 \ , \ \forall k \tag{3.12}$$

$$\sum_{k=1}^{n} g_c^k \leq h-1 \ , \ \forall c \tag{3.13}$$

The objective in (3.7) is to minimize the number of used links for each group. Constraint (3.8) says that if source k was participating in group c the outgoing flow from it must be equal to h in the time slot for that group. Constraint (3.9) forces z_{ij}^c to be equal to 1 if the flow on edge (i,j) was not 0. Constraint (3.10) says that in a certain group (i.e. at a certain time slot) the amount of flow (of all sources) entering a node equals the amount of flow leaving that node. Constraint (3.11) says that the sum of flow of all sources in a certain group cannot exceed the capacity of any link which is equal to $h-1$. Constraint (3.12) ensures that each source participates in one group only, and (3.13) guarantees that a group contains no more than $h-1$ sources.

This MILP guarantees fair bandwidth sharing, i.e., a source cannot transmit again unless all other sources have transmitted. This is ensured by constraint (3.12), which forces each source to participate in one group only. As we will show later in Section 3.7, a source might have the opportunity to transmit more than once without affecting the throughput of other

sources; we call this opportunistic transmission. The MILP can be modified for opportunistic transmissions as follows:

The objective function should be:

$$Minimize \ \sum_{c=1}^{M}(\sum_{k=1}^{n} g_c^k + \sum_{\forall(i,j)\in E} z_{ij}^c) \tag{3.14}$$

with the following modifications on constraints 3.12 and 3.13:

$$\sum_{c=1}^{M} g_c^k \geq 1 \ , \ \forall k \tag{3.15}$$

$$\sum_{k=1}^{n} g_c^k = h - 1 \ , \ \forall c \tag{3.16}$$

Now the bandwidth is utilized by constraint (3.16) that sets the size of all groups to its maximum size $h-1$, and a source is allowed to participate in more than one group by constraint (3.15).

3.6.4 Implementation

Assumption 8 states that a node can receive from, or transmit to, multiple nodes simultaneously. Practically, this can be through using multiple transceivers utilizing different frequency channels. On the other hand, if we want to remove this assumption completely, time scheduling of node transmissions can be used. It can be shown for some simple cases, that if D is the number of time slots, where D is a function of N and D increases as N increases, that the difference in D between $(1:N)$ protection and network coding protection will be very small. We elaborate more on scheduling in Chapter 4.

3.6.5 Decoding at the sink node

We showed in Section 3.5 that {0,1} coding can be accomplished in $O(n^2)$. In this section we discuss the decoding process at the sink node. We assume that each packet carries the coding vector for the combination in that packet. Which can be done by adding a bit-map of

length n in the header of each packet, where a 1 at position i indicates that the data unit from source i participates in this combination, since we use {0,1} coding.

Assuming a single-link failure, the sink is guaranteed to receive at least one trivial combination consisting of a single data unit (and at least two trivial combinations if no failure occurs), which does not need any further processing to retrieve the carried symbol. This is due to the fact that the coding tree will have at least two leaf nodes in L_s (when the tree is a path), which will produce these single data unit combinations; we refer to these combinations as *leaf combinations*. The details of decoding process were previously discussed in Section 3.5.1, where we showed how to solve the set of linear combinations at the sink using a simple procedure that runs in $O(n)$. Note that when a network has a limited cut h, the decoding time will be of order $O(h)$.

Clearly, it is better to have as many leaves as possible in the coding tree. This is because as the number of leaf combinations increases the running time will decrease. Unfortunately, this is an NP-complete problem and cannot be solved in polynomial time. Note that the coding tree algorithm does not reduce the number of leaf nodes in the initial tree (unless it is trimming unneeded leaves in L_s that will create duplicate trivial combinations), it just modifies the tree to ensure that all the leaf nodes are in L_s. Therefore, to maximize the number of leaf nodes in the coding tree we can start by finding the *maximum leaf spanning tree* on the bipartite graph containing U_s and L_s (which is a known NP-complete problem (50)), then run the coding tree algorithm to ensure that all leaf nodes are in L_s, which will transfer the maximum leaf spanning tree to the maximum leaf coding tree in polynomial time.

To keep the coding simple the initial tree must be constructed in an efficient manner. There are two well known tree search strategies, the DFS-tree (Depth First Search) and the BFS-tree (Breadth First Tree). To compare these two strategies, we ran our algorithm in Section 3.5 using both options to see which strategy performs better in terms of producing leaf combinations (leaf nodes). The results are shown in Figure 3.14, where each point on the graph corresponds to the average of 50 runs using the same number of nodes in U_s and choosing the number of L_s randomly between $|U_s| + 1$ and $2|U_s|$. From the figure, we can see

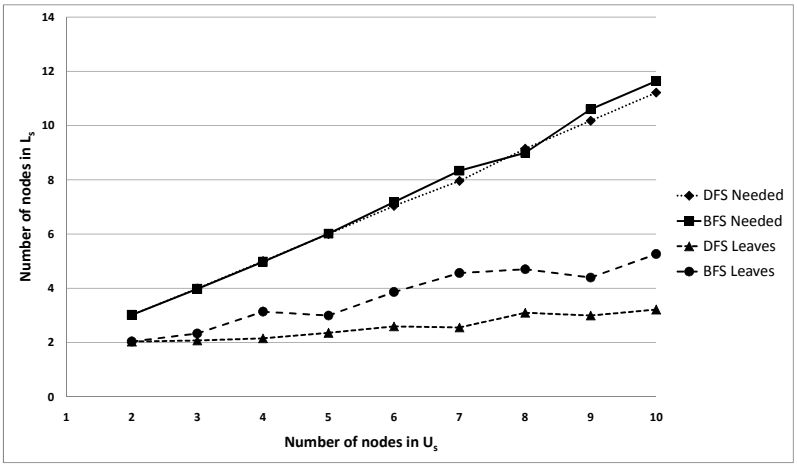


Figure 3.14 Comparing the number of produced leaf nodes for a BFS-tree and a DFS-tree

that on average BFS produces more leaf combinations than DFS. Although not optimal, in terms of producing maximum number of leaves, we use a BFS-tree as the initial tree in our algorithm for its simplicity where it can be constructed in $O(|E|)$ steps (49).

It is worth mentioning that the decoding process will be simplified if there was a node, say x, in L_s with a degree equal to n. This is because we can send n trivial combinations from nodes other than x, plus a combination carrying the XOR of all data units that is created at node x. An example is shown in Figure 3.9(a). If this was the case then decoding is only needed if a trivial combination was lost.

3.6.6 Sink to sources transmission

To send data to users, the sink can literally reverse the process used by the sources. That is, generate $n+1$ combinations, such that any n of them are linearly independent. Then, forward these combinations to the nodes in L_s so that they can collaboratively recover the original data units and send each of which to its designated receiver (user). One way to do this is by using the same coding vectors for the combinations received from the sources. For example, if the sink received the combination $b_i \oplus b_j$ on path k, then when sending data upstream to sources the data units for sources b_i and b_j are XORed and sent back on the same path k.

To recover the data units the users should collaborate with routers in the following manner.

The recovery can start from the L_s nodes that receive single data unit combinations, which correspond to leaf nodes in the coding tree (there are at least two such nodes). These nodes send the received data units to their corresponding users (each leaf node in L_s sends to a single source). By doing this the users have received the data units destined to them. After that, to continue the decoding process, the users send these data units to their other neighboring nodes in L_s on the coding tree (since each source has at least two neighbors in L_s). Upon receiving a data unit, a node in L_s removes this data unit from the combination it received from the sink. The L_s nodes continues to remove the received data units until the L_s node has received data units from all but one of its U_s neighbors, say y, after which the data unit for user y will be recovered, and then sent to user y. The process repeats until all the original data units are recovered by the user nodes.

The recovery will be successful as long as there is at least one leaf combination that was received successfully by a node in L_s. We are guaranteed to have such a leaf combination when a single-link failure takes place. Basically there are two possibilities:

1. The failure affects a leaf combination: in this case as we discussed above we are guaranteed to have at least one other leaf combination.

2. The failure affects a non-leaf combination: in this case the coding tree is divided into two smaller trees, each of which will have at least one leaf combination.

In the worst case, when the coding tree is a path, and one of the leaf combinations is lost due to a failure, this process takes $O(2n)$ transmissions or $O(2h)$ in networks with limited min-cuts.

3.7 Network Performance

Since redundant data is sent to provide protection, the effective data rate will be decreased, compared to the case when there is no protection. In this section, we will study the effect of network coding-based protection on the effective data rate. Specifically, we will discuss the following cases:

- **Case 1:** Fair bandwidth sharing, with no protection.
- **Case 2:** Fair bandwidth sharing, with protection.
- **Case 3:** Opportunistic transmission, with no protection.
- **Case 4:** Opportunistic transmission, with protection.

By fair bandwidth sharing we mean that a source does not transmit again until all other sources have transmitted, and by opportunistic transmission we mean that a source transmits whenever it has an opportunity to do so. In our discussion, we define the rate $\mathcal{R}$ as the number of data units that can be received by the sink per unit time. Also we assume the following:

1. Sources will be divided into groups, like we did in section 3.6, if h was not large enough to forward all the $n+1$ combinations in the case of coding, or the n data units when there is no protection.
2. There are two edge-disjoint paths from every source to the sink, i.e., minimum number of neighbors is used. This assumption will simplify the analysis, and will not affect its validity. This is because the rate is affected by the total number of combinations received at the sink, and the number of unique data symbols that can be recovered, but not by the number of occurrences of the data symbols in the combinations.
3. We assume that the selected groups are feasible as defined in section 3.6.

Case 1.a: When $h \geq n$, the sink can receive all the n data units at the same time, which means that:

$$\mathcal{R} = n$$

Case 1.b: When $h < n$, the sources should be grouped, which will cause the rate to vary at the sink depending on the way the sources were divided:

1. The best grouping scenario is when the sources are sorted in disjoint groups, giving rise to $\lceil \frac{n}{h} \rceil$ groups, i.e., $\lceil \frac{n}{h} \rceil$ time units are needed for all the sources to be covered, which

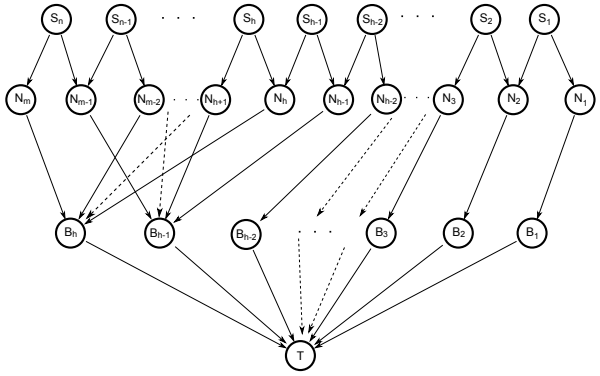


Figure 3.15 Worst case grouping

means that:

$$\mathcal{R} = \frac{n}{\lceil \frac{n}{h} \rceil} \leq \frac{n}{\frac{n}{h}} = h$$

2. Since every source has two-edge disjoint paths to the sink, then any source can reach two different edges in the min-cut. The worst grouping scenario occurs when there are $h-2$ min-cut edges that can be only reached by $h-2$ sources, forcing the remaining $n-h+2$ sources to use just the remaining two min-cut links. Because we assume no protection in this case, each source can use one edge in the min-cut, hence, the $n-h+2$ sources can be divided into $\lceil \frac{n-h+2}{2} \rceil$ pairs, each pair when combined with the other $h-2$ sources will form a group. Which produces $\lceil \frac{n-h+2}{2} \rceil$ similar groups that only differ in two elements. The network in Fig. 3.15 shows an example, where the sources S_1, S_2, ...,S_{h-2} are present in all the selected groups (although we assume that they transmit in only one time slot out of the $\lceil \frac{n-h+2}{2} \rceil$), and the sources from S_{h-1} to S_n can connect to the sink only through the last two min-cut links. In this case the $n-h+2$ sources will share these two links in $\lceil \frac{n-h+2}{2} \rceil$ time slots. Which means that:

$$\mathcal{R} = \frac{n}{\lceil \frac{n-h+2}{2} \rceil} \leq \frac{n}{\frac{n-h+2}{2}} = \frac{2n}{n-h+2}$$

It can be seen that the last two cases are equivalent when $h = 2$, and they both give $\mathcal{R} = 2$.

Case 2.a: When $h \geq n+1$ the sink can receive the $n+1$ combinations at the same time and recover all the n data units using any n combinations, that is:

$$\mathcal{R} = n$$

Case 2.b: When $h < n+1$ the sources should be divided into groups, and the rate at the sink will depend on how the grouping was accomplished:

1. As before, the best grouping is when the sources can be divided into disjoint groups, but in this case, since we assume protection is provided, the groups will be of size $h-1$, thus producing at most $\lceil \frac{n}{h-1} \rceil$ groups. Hence, the rate will be:

$$\mathcal{R} = \frac{n}{\lceil \frac{n}{h-1} \rceil} \leq \frac{n}{\frac{n}{h-1}} = h-1$$

2. The worst case grouping is similar to that discussed in Case 1.b, but in this case since protection is assumed, the $n-h+2$ sources mentioned earlier will not be divided into pairs, rather, each of which will be active alone with the other $h-2$ sources, thus giving rise to $n-h+2$ groups, each of size $h-1$. The network in Figure3.15 still gives a valid example. The rate in this case is:

$$\mathcal{R} = \frac{n}{n-h+2}$$

Note that the last two cases are also equivalent when $h = 2$, and give $\mathcal{R} = 1$, since each source must use two edge-disjoint paths to forward data to the sink.

Case 3: The calculations from Case 1 are still valid for this case, except when $h < n$ with worst case grouping, in which the rate at the sink will equal h, since the $h-2$ sources are allowed to transmit in every one of the $\lceil \frac{n-h+2}{2} \rceil$ time slots. To capture the difference, we should consider the rate at the sources, where it was $\frac{1}{\lceil \frac{n-h+2}{2} \rceil}$ for each of the n sources in Case 1. However, in this case, $\mathcal{R} = 1$ for each of the $h-2$ sources that are repeated in all groups and $\frac{1}{\lceil \frac{n-h+2}{2} \rceil}$ for each of the $n-h+2$ sources that can only use two min-cut links.

Case 4: The only difference between this case and case 2, is when $h < n + 1$ with worst case grouping, in which the rate at the sink will always be equal to $h - 1$. Again, to see the difference we should consider the rate at the sources, where it will be 1 for each of the $h - 2$ common sources, and $\frac{1}{n-h+2}$ for each of the $n - h + 2$ sources that share only two min-cut edges.

Note that by comparing the cases of no protection to the cases when protection is provided, i.e., compare case 1 to 2 and case 3 to 4, we can see that our protection scheme reduces the useful information rate by at most 50%. In such a case network coding-based protection will be equivalent to duplication-based or 1+1 protection.

3.8 Summary

In this chapter we presented a novel network coding-based approach that provides protection to many-to-one flows at the speed of proactive protection but at the cost of reactive protection. We derived and proved the necessary and sufficient conditions to achieve this protection for n source nodes. A polynomial-time algorithm was presented to perform the coding with {0,1} coefficients. We considered some of the practical issues related to our approach, such as adapting our scheme to general network topologies. Finally, we studied the effect of our network coding-based scheme on the performance of the network, where we showed that in the worst case our coding-based protection will act as 1+1 and reduce the useful information rate by at most 50%.

CHAPTER 4. Scheduling for reliable many-to-one flows

The previous chapter presented a proactive protection mechanism in Wireless Mesh Networks (WMNs), in which the data units of n users (sources) are combined (at the routers they are associated with) to produce $n+1$ linear combinations, such that any n of them are solvable. Digital Network Coding (DNC) is used to create these combinations, where the router nodes needs to know the bit representation of each of the data units to be coded. In this chapter, we consider the implementation of this protection scheme when all network nodes have single transceivers, and we solve the problem through a greedy algorithm that constructs a feasible schedule for the transmissions from the sources. We use this algorithm to compare the performance of our coding-based approach to the 1:N and the 1+1 protection schemes, in terms of the needed number of time slots. Furthermore, we study the scheduling problem, and we derive lower and upper bounds on the needed number of time slots in a DNC-based schedule. After that, we investigate the possible benefits that could be gained from using analog network coding (ANC) in the scheduling process. We show how ANC can make the scheduling more efficient (i.e., needs fewer time slots). We also discuss some special cases, in which the optimal (i.e., minimum) number of time slots can be achieved for an ANC-based schedule in polynomial time. Moreover, we show that, theoretically, ANC can outperform DNC by a factor of n if the graph induced by the user and router nodes had a certain structure.

4.1 Scheduling Based on Digital Network Coding

4.1.1 Greedy algorithm

In the previous chapter we assumed that a node can receive from, and transmit to, multiple nodes at the same time, which may be possible if a node has multiple transceivers. As men-

tioned in Subsection 3.6.4, if each node has a single transceiver, a scheduling mechanism for the sources transmissions should be used. In this section we introduce a greedy algorithm that constructs a feasible schedule for the transmissions of the sources in U_s (the set of users), taking into consideration their connectivity with their neighbors in L_s (the set of routers). We use this algorithm as a common ground to compare the performance of our network coding-based protection scheme with the $1:N$, and the $1+1$ protection schemes, in terms of the number of needed time slots. Before stating the algorithm, let us define the feasibility conditions of a schedule for the three protection schemes, assuming that the max-flow between L_s and the common destination is h:

For Network Coding-Based protection the following must hold:

- The number of sources in a certain slot does not exceed $h-1$.
- If source x is scheduled in a time slot, then no other source that has a common neighbor with x can be scheduled in the same time slot.

For 1:N protection the following must hold:

- The number of sources in a certain slot does not exceed $h-1$.
- If source x is scheduled in a time slot, then there must be at least *one* neighbor for x, which does not receive from any source other than x in that slot.

For 1+1 protection the following must hold:

- The number of sources in a certain slot does not exceed $h/2$.
- If source x is scheduled in a time slot, then there must be at least *two* neighbors for x, which do not receive from any source other than x in that slot.

Taking the feasibility conditions into account, Algorithm 3 shows how to build a schedule that satisfies these conditions for the case of network coding-based protection (we will discuss the other two cases shortly). For each time slot, the algorithm selects the source with the least degree in U_s. Then, it excludes all the sources that will violate the feasibility conditions from future choices, by putting them in the *Colliding_Sources* set. The previous two

steps are repeated until no more sources can be added to the current time slot. After that, *Colliding_Sources* is reinitialized and the algorithm starts filling the next time slot. The process repeats until all sources are scheduled.

Algorithm 3 Scheduling Algorithm

1: //Defining Variables
2: $SchdSet = \emptyset$; //a set that contains the scheduled sources
3: $Colliding\_Sources[|U_s|] = \emptyset$; //a set that contains the sources that will collide with the scheduled source if both transmit at the same time
4: $Schedule[|U_s|][|U_s|]$; //transmissions schedule
5: $Slot = 0$; //the number of needed time slots
6: $Index = 0$; //source index in a time slot
7: h = Max-flow; //the max-flow from L_s to the sink
8: $Found$ = TRUE; //a boolean variable, which is TRUE if a new source is added
9: **while** ($|SchdSet| < |U_s|$) **do**
10: $x = \emptyset$;
11: **if** ($Index > h - 1$ || $Found ==$ FALSE) **then**
12: $Slot$++;
13: $Index = 0$;
14: $Colliding\_Sources[|U_s|] = \emptyset$;
15: **end if**
16: x = Select the node in L_s with the least degree, say u,such that ($u \notin Colliding\_Sources$) **AND** ($u \notin SchdSet$);
17: **if** ($x == \emptyset$) **then**
18: $Found$ = FALSE;
19: **else**
20: $SchdSet = SchedSet \cup x$;
21: $Schedule[Slot][Index] = x$;
22: $Colliding\_Sources = Colliding\_Sources \cup$ {All the U_s neighbors of the L_s neighbors of x that are not in $SchdSet$}
23: $Index$++;
24: **end if**
25: **end while**

For the case of $1 : N$ protection, one slight modification is needed. That is, in step 22 *Colliding_Sources* should be modified to include *All the U_s neighbors of the one node in L_s that is a neighbor to x, and has the least degree.* The node with least degree is chosen to reduce the number of conflicting sources, which enables us to put more sources in the same time slot.

However, in the case of $1 + 1$ protection, two modifications are required:

1. The condition of the **if** statement in 11 should be modified to ($index > h/2$ || $Found$ == FALSE).

2. In step 22, *Colliding_Sources* should be modified to include *All the U_s neighbors of the*

two nodes in L_s that are neighbors to x, and have the least degrees.

To be as practical as possible in our comparison, the sources in U_s should have small degrees. This is because 1) the sources only see the nearby neighbors (routers), which fall within their vicinity, and 2) the routers in reality are not placed too close to each other, so that the maximum amount of users are covered with the minimum number of routers. In addition to the source degree, the max-flow between L_s and the sink should also be small, since in reality the network gets narrower as we approach the sink.

We compared the three schemes based on the algorithm and the following setup. The cardinality of U_s was varied from 2 to 20. In each step, we generated 10 different topologies, with the cardinality of L_s being randomly chosen between $|U_s| + 1$ and $2|U_s|$, and with connectivity conditions in Lemma 1 satisfied. The algorithm was then executed for the three protection schemes on each of the ten topologies, and the average was taken.

We conducted two experiments with two different values for the max-flow. Specifically, we made h equals $(L_s/4) + 1$ and $(L_s/6) + 1$, the results are shown in Figures 4.1 and 4.2 respectively. It can be seen that the difference between 1:N and network coding is very small, and it shrinks further as the max-flow decreases, i.e., in more practical cases. Obviously, 1+1 protection performance is poor compared to the other two schemes, since it approximately consumes twice the resources used by the 1:N or the network coding-based protection. Thus, fewer sources can be scheduled together.

These results compare the performance of the three schemes when no failure occurs, which is unjust to the 1+1 and network coding-based protection schemes when compared to the 1:N protection. This is because, in the case of a failure, the performance of 1+1 and network coding will not be affected, i.e., no more time slots are required. On the other hand, the performance of 1:N will get worse, because it will consume more time slots to do the rescheduling, which establishes the advantage of network coding-based protection over the other two schemes.

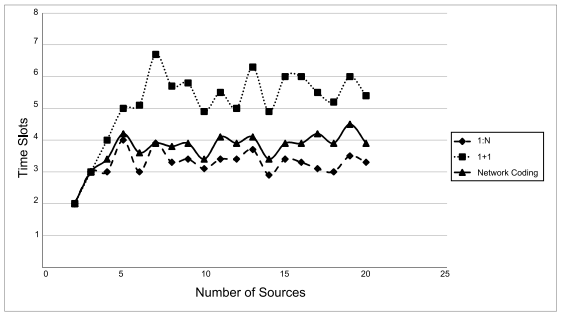


Figure 4.1 Max-flow=$(L_s/4)+1$

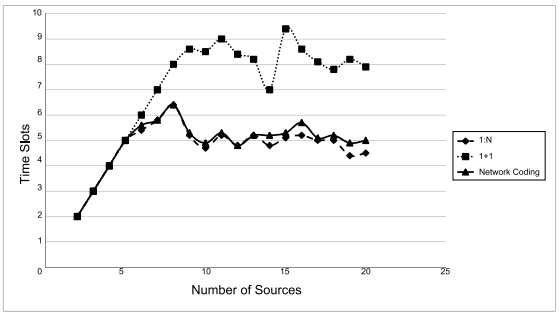


Figure 4.2 Max-flow=$(L_s/6)+1$

4.1.2 Bounds on schedule length

In a feasible schedule for DNC, users in U_s that have a common router in L_s (i.e., connected to the same router) cannot be scheduled in the same time slot (this is called the *feasibility condition* for a DNC-based schedule, which was discussed in the previous section and will be referred to later in Section 4.3). This is because the router needs to know the actual bit representation for each of the data units from the two users to be able to perform digital network coding. That is, we say that there is a conflict between two users in U_s if they have a common neighboring router in L_s. In this case there is a direct relationship between DNC-based scheduling and the vertex coloring problem. The DNC-based scheduling problem on some bipartite graph $G(V, E)$, where $V = U_s \cup L_s$, can be shown to be NP-Complete by the following reduction from the vertex-coloring problem on a given conflict graph $H(V', E')$:

- The nodes in V' are mapped to the nodes in U_s.
- An edge (u, v) in H is transformed to a node n_{uv} that belongs to L_s, and is connected to nodes u and v in U_s.
- For each node in U_s add a neighbor in L_s and connect them together to guarantee that any k nodes in U_s are connected to at least $k+1$ node in L_s.

Thus an optimal and feasible schedule of k time slots exists if and only if H is k-chromatic. This mapping is not the only possible mapping from the vertex coloring problem to the DNC-based scheduling problem. However, this does not invalidate the reduction. Also note that by

using this transformation, all the nodes in L_s will have degree 2. Nevertheless, this does not mean that we can construct a conflict graph for G only if all of its L_s nodes had degree 2. For example consider the graph in Figure 4.3, applying the transformation stated above will result in the graph shown in Figure 4.5. However, this is not the only graph that has a conflict graph similar to H in Figure 4.3. For instance the graph shown in Figure 4.6 will have exactly the same conflict graph H, and thus solving the vertex-coloring on H provides a feasible and optimal schedule for both the graphs in Figures 4.5 and 4.6. Although in our coming discussion we use examples where all the nodes in L_s have degree 2, the analysis is valid and applies for any graph G.

We use this relation with the vertex-coloring problem to bound the number of needed time slots for a DNC-based schedule. Let Δ_{L_s} be the maximum node degree in L_s in G, let T_{schd} be the minimum number of time slots needed for a schedule (or equivalently the number of colors in H), and let Δ_H be the maximum node degree in the conflict graph H corresponding to G. We have the following bounds on the number of needed time slots.

$$\Delta_{L_s} \leq T_{schd} \leq \Delta_H$$

The lower bound follows since no two sources in U_s can be scheduled together if they have a common neighbor in L_s. The upper bound follows from Brook's theorem (1), which says that *"for a connected graph H that is neither a full graph nor an odd cycle the chromatic number $\chi(H)$ is less than or equal to the maximum node degree Δ in H"*. If H is a full graph or an odd cycle then $\chi(H) = \Delta + 1$.

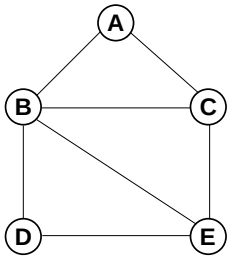


Figure 4.3 Graph H

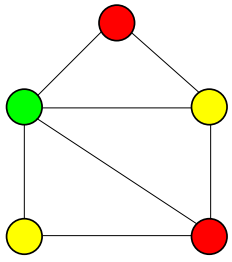


Figure 4.4 3-coloring of H

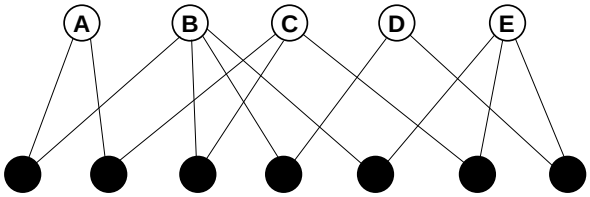


Figure 4.5 Corresponding Graph G

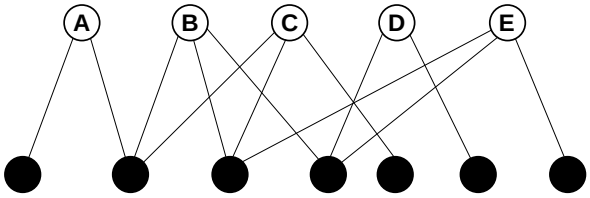


Figure 4.6 An alternative corresponding Graph G

It easy to solve the coloring problem for graph H in Figure 4.3 by just looking at it. The minimum number of needed colors for H is 3 and the solution is shown in Figure 4.4. This indicates that an optimal schedule for the transmissions of the sources in G needs 3 time slots. For this example note that $\Delta_{L_s} = 2$, $T_{schd} = 3$, and $\Delta_H = 4$.

Finally, one should note that creating the combinations in the L_s nodes is independent from the transmission schedule of the users. This is because, after all users have transmitted, a coding tree can be constructed to decide on the linear combinations. One possible coding tree for the graph G in Figure 4.5 is shown in Figure 4.7. The solid links represent the links on the tree, which will decide on the combinations as shown.

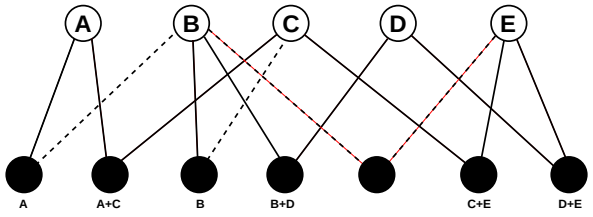


Figure 4.7 Coding tree on graph G

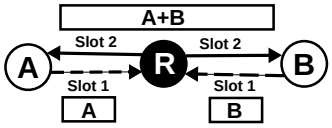


Figure 4.8 2-way relay channel

4.2 Scheduling based on Analog Network Coding

Analog network coding (ANC) was proposed in (3) to enhance the capacity of the two-way relay channel, where there are two terminal nodes that communicate with each other through a relay node. In ANC, if the two terminal nodes transmit together, the relay node receives their added analog signals and forwards the result to the two terminal nodes, instead of treating it as collision and discarding the received signal. After that, each terminal can recover the data destined to it by subtracting its data from the received signal. That is, with ANC two packets can be exchanged through the relay node and received by the terminals in only two time slots (compared to three with digital network coding or four without coding). Consider the example shown in Figure 4.8. There are two terminal nodes A and B, which need to exchange their data units. However, nodes A and B are not within the transmission ranges of each other. Therefore, ANC can be used to exchange their data units through the relay node R.

Note that in a many-to-one flow the sources in U_s are not exchanging data units through the routers in L_s, rather they are sending their data units to the routers in L_s, which in turn create combinations from the received data units and send them to the common destination T. These combinations can be created using digital network coding (DNC) or analog network coding. If DNC is used, a router cannot receive from more than a single source at a time as discussed in the previous section. However, if ANC is used, a router can receive from two sources at the same time. Therefore, a transmission schedule based on ANC may have a fewer number of time slots, when compared to a transmission schedule based on DNC. When the common destination receives all the combinations created by the nodes in L_s (using DNC or ANC), it will be able to decode them and recover the original data units.

Using ANC enables a node in L_s to receive from at most two sources at the same time. This means we can schedule transmissions from 2 nodes in U_s to the same node in L_s in the same time slot. Therefore, the lower bound on the minimum number of time slots needed, T_{schd}, is as follows:

$$\lceil \frac{\Delta_{L_s}}{2} \rceil \leq T_{schd}$$

Unfortunately, there is no clear relation between scheduling in this case and vertex coloring. However, we can still take Δ_H as a pessimistic upper bound. Note that unlike DNC-based scheduling, a node in L_s can receive from 2 nodes in U_s. Therefore, the transmissions from the users in a time slot will decide the combinations that will be sent to the common destination (because the signal will be physically added and cannot be changed). For example, if in a certain time slot both users A and B transmitted to a common router, the router will receive the sum of the two analog signals and will send it as is to the common destination. This sum will be treated as a degree-2 combination (i.e., a combination of 2 data units).

In addition to not hearing from more than two sources, there is one more constraint that must be satisfied in each time slot of an ANC-based schedule, and that is to have the suitable number of leaf combinations that will enable the decoding at the sink. Specifically, every set of sources that transmit in a certain time slot must produce at least two leaf combinations (note that this is not required in DNC). For example, Figure 4.9 shows a feasible schedule based on analog network coding that takes 2 time slots for graph G shown previously in Figure 4.5. In the first time slot, sources A and B transmit and produce 4 leaf combinations. The remaining sources transmit in the second slot and produce 4 leaf combinations also.

If duplicate or excess combinations are created by the transmissions in a time slot of an ANC-based schedule we assume that the duplicates are eliminated to reduce the number of packets sent to the destination. This can be done by finding a coding tree for the sources in each time slot and then discarding the combinations created at nodes not on the tree. For example in Figure 4.9 we only need the combinations inside the boxes, and all other combinations are discarded.

To summarize, in a feasible ANC-based schedule the following conditions must hold:

1. In any time slot a node in L_s does not receive from more than 2 nodes in U_s.

2. A set of sources that transmit in a certain time slot must produce at least 2 leaf combinations.

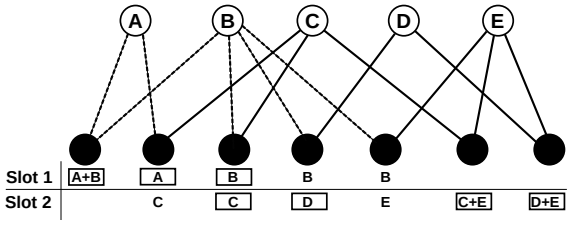


Figure 4.9 ANC Schedule

4.2.1 Special Case: When $\Delta_{L_s} = 2$

In this case, the first feasibility condition is satisfied by the construction of the bipartite graph G. However, we still need to satisfy the second feasibility condition for an ANC-based schedule. Practically, there are two possibilities:

1. There are 2 nodes in L_s with degree 1: in this case the second condition will also be satisfied and all sources in U_s can transmit in a single time slot.

2. All nodes in L_s have degree 2: in this case we can schedule any single node in U_s that has at least 2 neighbors in L_s in one time slot, and schedule all remaining users in U_s in the second time slot. Scheduling the node in the first time slot creates a sufficient number of leaf combinations in the second time slot to make it feasible.

4.2.2 Special Case: When G is a Tree

In this case the second feasibility condition is always satisfied for any set of sources, and we need to guarantee the first feasibility condition if the max-degree in L_s is larger than 2. We show that for this special case the needed number of time slots is $\lceil \frac{\Delta_{L_s}}{2} \rceil$.

Algorithm 4 ANC-Based Scheduling on a Tree

Input: Graph $G(V, E)$, $V = \{U_s \cup L_s\}$. G is bipartite, and a tree.
Output: Feasible ANC-based schedule that takes $\lceil \frac{\Delta_{L_s}}{2} \rceil$ time slots.

$S = \emptyset$, $N = \emptyset$, $T = \emptyset$, $x = 0$
while ($|T| < |U_s|$) **do**
  v = Node in L_s with max degree.
  Mark any two U_s neighbors of v for transmission.
  $S = S \cup$ {Marked neighbors of v}
  Delete all other U_s neighbors of v. //Temporarily
  $N = N \cup$ {All 2-hop neighbors of v}
  while ($N \neq \emptyset$) **do**
    Remove u from N
    if (u has a marked neighbor) **then**
      Mark an extra neighbor of u, & delete all others.
    else if (u has a deleted neighbor & remaining degree ≥ 2) **then**
      Mark two neighbors of u, & delete all others neighbors.
    else if (u has a deleted neighbor & remaining degree == 1) **then**
      Delete remaining neighbor
    end if
    $S = S \cup$ {Marked neighbors of u}
    $N = N \cup$ {All 2-hop neighbors of u not in S}
  end while
  Put all nodes in S in slot number x.
  Remove the nodes in S and their incident edges from G
  $T = T \cup S$, & $S = \emptyset$
  $x++$
end while

Let us consider Algorithm 4. In a certain time slot, the algorithm starts by considering the node in L_s with the highest degree (node v), it marks 2 of its neighbors (which are users in U_s), and deletes all other unmarked neighbors. After that, it considers other nodes in L_s with marked or deleted U_s neighbors. If a node , u, has a marked neighbor it can mark another one (since it can receive from at 2 users). If u has a deleted neighbor, and at least 2 remaining neighbors it marks 2 for transmission. Finally, if u has a deleted neighbor, and a single remaining neighbor it just deletes this remaining neighbor so that it will be scheduled later. When there are no more nodes to consider, time slot x is filled with the nodes in set S. To find a feasible schedule, the process is repeated until all users are scheduled. We now prove the following lemma, which will help us in proving that the needed number of time slots is $\lceil \frac{\Delta_{L_s}}{2} \rceil$.

Lemma 3. *If G is a tree and Algorithm 4 is used, then any node in L_s with a degree > 2 will have its degree reduced by 2 after each iteration of Algorithm 4.*

Proof. First of all, note that for an L_s node to have its degree reduced by 2, two of its neighbors in U_s must be marked. After marking 2 neighbors of node v (the one with the max degree in L_s) and deleting the remaining neighbors, node v starts reaching-out to its 2-hop neighbors (a 2-hop neighbor is reached when it has a marked or a deleted U_s neighbor), and in the next iteration the 2-hop neighbors will reach-out to the 4-hop neighbors, and so on.

We now prove the implication by contradiction. Assume Algorithm 4 was used, but some node, u, in L_s with degree > 2 did not have its degree reduced by 2 after running a single iteration of Algorithm 4. For this to happen, node u must have had all of its neighbors deleted in the reaching-out process, which means that all of its neighbors can be reached from node v (i.e., there are multiple paths from v to u). However, this contradicts the assumption that G is a tree. Therefore, every node in L_s with degree > 2 must have its degree reduced by 2 after Algorithm 4 is executed.

□

We are now ready to prove our claim.

Theorem 5. *If the bipartite graph G is a tree, a feasible ANC-based schedule can be achieved in exactly* $\lceil \frac{\Delta_{L_s}}{2} \rceil$ *time slots.*

Proof. A direct proof is used. We rely on Lemma 1 and the special case in Section 4.2.1 to prove this theorem. Basically, we need to count the number of time slots needed to reduce the max-degree in L_s to 2, let us call this number x. When the max degree in L_s is 2 all nodes in U_s will be marked by Algorithm 4, and non of them will be deleted. Therefore, the total number of needed time slots is $x + 1$. From the previous lemma the maximum degree after x time slots, d_x, is:

$$d_x = \Delta_{L_s} - 2x \tag{4.1}$$

Now let $d_x = 2$, and solve for x

$$x = \frac{\Delta_{L_s}}{2} - 1$$

Adding the last time slot for the remaining sources, and taking the ceiling of $\frac{\Delta_{L_s}}{2}$ if Δ_{L_s} was odd, we have:

$$T_{schd} = \lceil \frac{\Delta_{L_s}}{2} \rceil$$

□

4.2.3 Maximum Gain of ANC-Based Scheduling

The gain of ANC-based scheduling is maximized when it requires the minimum number of slots, while DNC-based scheduling requires the maximum number of slots. This happens when the following is true: 1) the maximum degree in L_s is 2 and there are at least two nodes in L_s with degree 1, i.e., only one time slot is needed for ANC. 2) the conflict graph (H) corresponding to G, is fully connected. In this case the gain is n. However, if all the nodes in L_s had degree 2, the gain will be $\frac{n}{2}$. An example is shown in Figure 4.10, where the maximum degree in L_s in the graph is 2. An ANC schedule is shown in the figure, where node A transmits in the first time slot, and all remaining nodes transmit in the second time slot. The digital

network coding based schedule is better shown on the corresponding conflict graph H, where each color on H represents a time slot. This is shown in Figure 4.11.

Recall that this scheduling problem has originated from the need to provide protection against a single failure for n data units. Therefore, practically, we do not need more than $2n$ neighboring routers. Therefor, although this case is theoretically possible and illustrates the maximum gain of ANC-based scheduling, it is practically unlikely to happen. This is because we need $\binom{n}{2}$ nodes in L_s to get a fully connected conflict graph

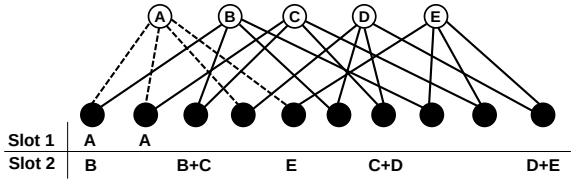


Figure 4.10 2-slot ANC Schedule

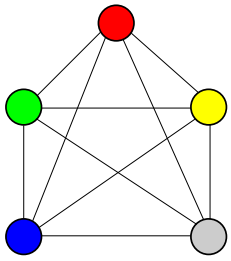


Figure 4.11 n-slot DNC Schedule

4.3 Performance Evaluation

In this section we compare the performance of ANC-based scheduling to DNC-based scheduling on random bipartite graphs, where we evaluate the gain of ANC-based scheduling over DNC-based scheduling. For this purpose, we use the simple greedy algorithm, shown in Algorithm 5.

In a certain time slot, the algorithm starts by selecting the user in U_s with the least degree (to reduce future conflicts with other users) that is not scheduled for transmission in any previous time slot. After that, all the users that will invalidate the schedule feasibility, are

Algorithm 5 Scheduling Algorithm

Input: Graph $G(V, E)$, $V = \{U_s \cup L_s\}$. G is bipartite.
Output: Feasible ANC-based or DNC-based schedule //depending on the feasibility conditions checked in Step 8

1: $S = \emptyset$, $F = \emptyset$, $N = U_s$
2: $Slots = 0$
3: **while** ($|S| < |U_s|$) **do**
4: Select user, u, with minimum degree in N, and $u \notin S$.
5: $S = S \cup \{u\}$
6: $F = F \cup \{All\ sources\ infeasible\ with\ any\ source\ in\ S\}$
7: $N = N \backslash F$
8: **if** ($N == \emptyset$) **then**
9: $Slots++$
10: $N = U_s$, & $F = \emptyset$
11: **end if**
12: **end while**
13: **return** $Slots$

removed by putting them in the set of forbidden sources F. This process is repeated until no more sources can be added to the current time slot. Then the algorithm starts filling a new time slot with users that are not scheduled previously, and so on. The infeasible sources in step 8 in algorithm 5, are the ones that will result in violating the feasibility conditions as defined earlier.

We ran Algorithm 5 on four different configurations, where the degree, D, of all nodes in U_s took the values 2, 3, 5 and 10. The results of the comparison for $D = 2$, $D = 3$, $D = 5$ and $D = 10$ are shown in Figures 4.12(a), 4.12(b), 4.12(c) and 4.12(d) respectively. For $D = 2$ the number of needed time slots is reduced by 42% on average if ANC is used instead of DNC on the same graph, with a maximum reduction of 50% at $L_s = 2$. For $D = 3$, the performance is slightly better, where the number of time slots is reduced by 43% on average, with a maximum reduction of 55% at $L_s = 8$. The same reduction (43%) on average is obtained for the case of $D = 5$, with a maximum reduction of 56% at $L_s = 10$. Finally, for $D = 10$, the number of time slots is reduced by 36% on average, with the maximum reduction of 54% at $L_s = 18$. Note that, the number of nodes in L_s at which the maximum reduction in the number of needed time slots occurs increases as the degree of the nodes in L_s increases. This is shown in Figure 4.13, which plots the ANC gain in each scenario against the number of nodes in L_s. When $|U_s| \leq 5$ the largest ANC gain is for the configuration with smallest degree. This is because

when the number of nodes is small, having a small degree will help in satisfying the second feasibility condition for ANC-based scheduling, and thus more users can be grouped together. However, when $|U_s| \geq 12$, the best gain is for the configuration with the largest degree. This is because, if the number of nodes is small and the node degree is large then DNC's performance will be very close to ANC (as shown in Figure 4.12(d) for $U_s \leq 6$). But, as the number of nodes increases, the performance of DNC will get worse faster than ANC because the conflicts will increase, and thus ANC starts to gain performance over DNC. Although we have shown that ANC can outperform DNC by a factor of n theoretically, it is clear that on randomly generated graphs the actual gain of ANC is around 2.

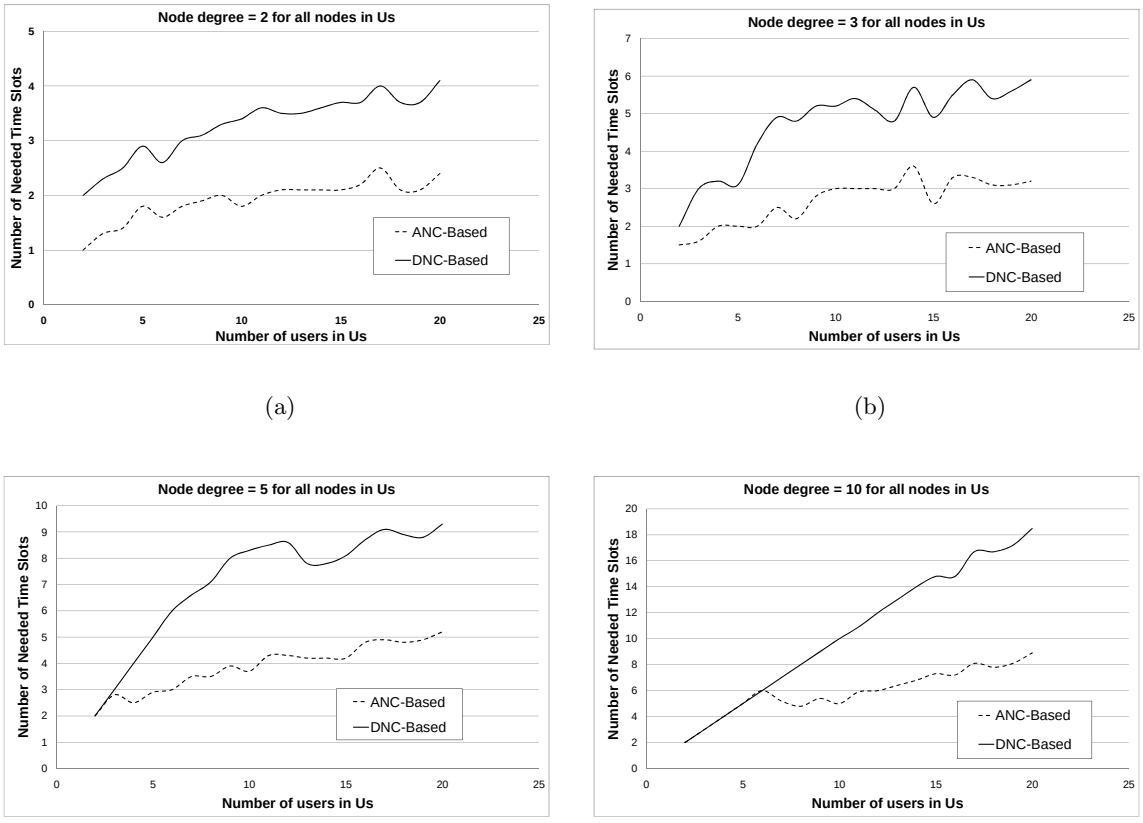


Figure 4.12 The performance of ANC-based scheduling is compared to the performance of DNC-based scheduling in four different settings (a) Degree = 2 , (b) Degree = 3 (c) Degree = 5 (d) Degree = 10

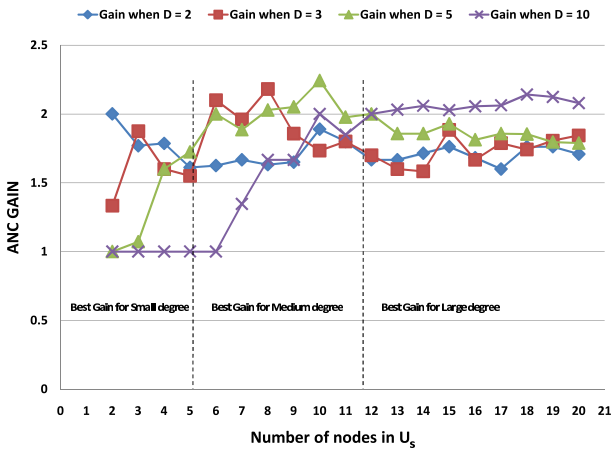


Figure 4.13 ANC Gain comparison for different node degree

4.4 Summary

The implementation of the protection mechanism presented in the previous chapter was studied, when each network node has a single transceiver. This was done through a scheduling algorithm, which showed that the performance of our approach in terms of the number of needed time slots, and hence throughput, is comparable to the performance of 1:N when no failures occur, and is better in the case of failures. In addition, we derived upper and lower bounds on the needed number of time slots in a schedule based on digital network coding (DNC). After that, we studied the scheduling problem using analog network coding (ANC). Applying ANC reduces the number of needed time slots to schedule the transmissions of the data units from the sources to the coding nodes. We showed some cases in which the optimal ANC-based schedule can be found in polynomial time. In addition, we showed that using ANC can reduce the number of required time slots by at most a factor of n compared to using DNC, where n is the number of source nodes. This however, is highly unlikely to happen in practice. We evaluated and compared the scheduling problem based on both DNC and ANC on randomly generated graphs. It was shown that the gain from ANC is usually around 2, and that the ANC gain depends on the node degree of the sources. Specifically, the more sources

we have the more dense the network should be, and vice versa.

CHAPTER 5. Scalable redundancy for Sensors-to-Sink communication in Wireless Sensor Networks

5.1 Introduction

In the previous chapter we showed that network coding can be used to tolerate a single link failure in a many-to-one flow, by having each set of n sources send at least $n+1$ combinations to the sink on $n+1$ link-disjoint paths, such that any n of them are linearly independent. However, this scheme is centralized and requires global information (in the first run at least). In addition, the scheme in the previous chapter assumes a fixed set of sources at a certain time, which may not be suitable for WSNs.

In this chapter, we introduce a new technique that will dramatically reduce the redundancy overhead compared with that in duplication-based multipath mechanisms, and yet still achieves the same level of survivability and recovery speed. In Section 5.2, we start by showing how to use deterministic network coding in a distributed manner to protect the data units from k sources against a single packet loss, by sending $k+1$ combinations to the base station (BS) such that any k of them are solvable. We extended our solution in Section 5.3 to tolerate multiple packet losses. Moreover, we discuss some coding/decoding-related issues in Section 5.4. We introduce relative indexing, which reduces the overhead of forwarding the coding vectors. Also, we show that the coding scheme allows partial recovery of the data units if more than one packet is lost, given that some conditions are satisfied. To use our technique to tolerate node or link failures, we need each of the combinations to use a path that is disjoint from paths used by all other combinations. Therefore, we introduce a simple routing protocol that can produce maximally disjoint paths in Section 5.5. In Section 5.6, we study the relationship between the probability of successful recovery of all k data units at the BS, and the number of sources

protected together taking into consideration their hop distance from the BS. From this study we can decide on the appropriate number of sources to be protected together, so that the probability of successful recovery is higher than a certain threshold. Simulation results are presented in Section 5.7, and they show that our scheme is highly scalable and performs better as the number of sources increases.

5.2 Operation

We consider a dense and uniformly distributed WSN. Assuming that at most one packet can be lost, our objective is to practically provide proactive protection to the information flow from the sensors to the BS using as few resources as possible. Before discussing the details of node operation, we need to clarify our assumptions and notations. Specifically, we assume the following:

1. We assume a wireless sensor network in which the sensor nodes can be organized into levels or rings around the base station, such that, the minimum hop count between the sensor nodes in ring i and the BS is i hops. An example is shown in Figure 5.1, where nodes d, e and f are in the first ring, nodes c and b are in the second ring and finally node a is in the third ring around the BS.

2. We assume that there are $\mathcal{R}$ levels in the sensor network, and we denote the level of node u by l_u.

3. Routing a packet ensures its progress towards the sink in each transmission, i.e. a packet gets closer to the sink by one hop after each transmission.

4. Sensors generate data periodically and at the same rate.

5. The data units for all sensor nodes are equal in size, and the data unit for sensor node u is denoted by d_u.

6. We use p to denote a certain packet, and we reserve d_p to only represent the information symbol carried in packet p.

Assume that sensor node u has a data unit d_u that must be forwarded to the BS. Node u can send two copies of d_u to the sink to tolerate a single packet loss as a proactive alternative to retransmission. However, if there is a large number of sensor nodes that need to forward information to the sink, this solution is not efficient anymore. This is because 50% of the forwarded information is redundant, and thus, at least 50% of the network resources (bandwidth and energy) will be wasted to provide this redundancy.

Assume that node u has sent two copies of d_u to the sink. Naturally, this is done through multi-hop communication for each of these packets. Suppose that as the two packets are forwarded, $k-1$ of the forwarding nodes had data units of their own that also need to be sent to the BS. We show how to protect d_u and the $k-1$ other data units by forwarding only $k+1$ packets, through the use of network coding. In general, the nodes in a wireless sensor network can be divided into the following three types:

1. **Type 1.** A source with no data to relay.
2. **Type 2.** A source with data to relay.
3. **Type 3.** Just a relay with no data of its own.

To tolerate a single loss using network coding in a distributed manner, we need to specify the way a node operates given its local information. We define the following operation for each class of nodes:

1. **Type 1.** Assume node u only has its own data unit d_u, and has no packets from other sensors to relay. Then node u just sends two copies of d_u to be able to tolerate a single loss. For example, node a in Figure 5.1. Since wireless communication is used, these two copies can be delivered to the next hop nodes in one transmission.
2. **Type 2.** Assume node u has its own data unit d_u, and in addition has received another packet, say p, which needs to be relayed (e.g., node b in Figure 5.1). Then node u produces the following two packets:
   (a) **Packet 1.** Contains the bitwise XOR of d_u and d_p.

(b) **Packet 2.** Contains only d_u.

3. **Type 3.** Assume node u has no data of its own, and has received a packet p that it needs to relay. Then it just forwards p as is, e.g., nodes c, d, e and f in Figure 5.1.

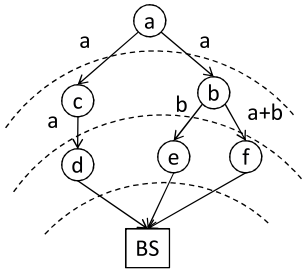


Figure 5.1 Protecting data from two sources against a single packet loss

The type of a node changes according to its status, i.e., if node u is of Type 3 at a certain time instant t_i, it might become a Type 1 or Type 2 node at t_{i+1} if a data unit is generated locally. Now recall the scenario of node u and the $k-1$ other sources that forward the data units of u. In this scenario node u is the only node of Type 1, and the $k-1$ sources are of Type 2. Note also, that each of the $k-1$ sources increases the number of total packets by 1, which means that the total number of generated packets is $k+1$. Let us call this operation a *forwarding process initiated by node* u, or for short, an *F-process* initiated by a Type 1 node, u.

We now prove that by following the rules of operation, if k sources are involved in a certain F-process then they will produce $k+1$ packets such that any k of them carry a solvable set of combinations.

Theorem 6. *If s is the k^{th} source to participate in an F-process initiated by node u, in which the number of involved sources until now is $k-1$ (including u), then by following the rules of operation described above, s will increase the number of packets to $k+1$, such that any k packets from them carry a solvable set of combinations.*

Proof. We prove this theorem by induction. We need to show that if any $k-1$ from the current k combinations (produced by the $k-1$ sources) are solvable, then the participation of the k^{th} source will produce $k+1$ combinations such that any k of them are solvable.

The *basis step* is when the F-process is first initiated at node u. By following the rules of operation, node u will send two copies of d_u. Thus we have $k = 1$ source, and $k + 1 = 2$ combinations (trivial combinations in this case), such that any one (k) of them is solvable, which is obvious since each combination contains only one data unit, so if one packet is lost the other is sufficient to recover d_u.

To prove the *inductive step*, assume that $k - 1$ sources have participated and any $k - 1$ from the current k combinations are solvable. Node s can participate only if it is of Type 2, i.e., it received one packet (say p) from the current k packets and it has its own data unit d_s that must be sent to the BS. By following the rules, s will produce two packets: 1) a packet containing the XOR of d_s with d_p (note that the number of total packets is still k since p and d_u are merged into one packet), and 2) a packet containing d_s (i.e., the number of packets is increased by 1). We now need to show that losing either one of the newly created packets will leave us with k solvable combinations. If we lose the first packet, then the second packet will be sufficient to recover d_s, and from our assumptions the remaining $k - 1$ combinations will be sufficient to recover the remaining $k - 1$ data units. If we lose the second packet, then from our assumptions we can recover all the data units in d_p (and thus d_p itself) using the $k - 1$ combinations other than $d_s \oplus d_p$, and then recover d_s by $d_p \oplus (d_s \oplus d_p)$. Finally, if we lose a packet other than those produced by s, we can recover d_s from the second packet produced by s, which leaves us with $k - 1$ solvable combinations in $k - 1$ unknowns. ■

A node of Type 1, will initiate a process that will involve a large number of nodes of the other two types. To distinguish the packets belonging to the process initiated by a node of Type 1, we add a new field in all the packets generated by this process. Let us call this field the *"initiator ID"* or *IID* for short. A node of Type 1 will put its ID and a time stamp in this field, and a node of Type 2 (say u) that XORs d_p with its data unit will copy the IID from p and put it in both generated packets. If a Type 2 node receives 2 or more packets with the same IID it must not combine its data with more than one of them, since this may produce dependent combinations.

Let P_i^j be the set of packets leaving level i (i.e., generated by level i, or just forwarded by it) that belong to the process initiated by node j, then $|P_i^j|$ is equal to:

$$|P_i^j| = 1 + \sum_{k=i}^{l_j} S_k^j = 1 + N_i^{l_j} \tag{5.1}$$

where S_k^j is the number of source nodes (Type 2) in level k that are involved in the process initiated by j, and N_i^j is the number of source nodes in the levels from i to l_j that are involved in the process initiated by node j. Of course, there is only one node of Type 1 in the process initiated by node j, and that is j itself.

There should be a limit on the number of combinations that can be produced by an F-process. Let m denote the maximum number of combinations that can be produced in an F-process, then m should not exceed the network min-cut. Note that the min-cut is not the only thing that can limit the number of combinations produced by an F-process. However, for now we take m to be the min-cut, and we elaborate more on the selection of m later. In a sufficiently dense and uniformly distributed WSN, the minimum cut is usually the number of one hop neighbors of the BS. Note also that since the maximum number of combinations is m, then no more than $m-1$ sources can be involved in any F-process. To insure this, we add a new field in the packet format; Let us call it *"Number of Remaining Combinations"* or *"NRC"* for short. We now redefine the operation of Type 1 and Type 2 nodes as follows (the operation of Type 3 nodes do not change):

1. **Type 1.** Assume node u only has its own data unit d_u, and has no data units from other sensors to relay. Then node u just sends two copies of d_u, and makes $IID = u$ and $NRC = \lfloor m/2 \rfloor$ in both packets.

2. **Type 2.** Assume node u has its own data unit d_u, and in addition has received another packet, say p, that it needs to relay. The operation of u depends on the NRC value of p as follows:

   (a) If $NRC(p) \geq 2$ node u produces the following two packets, and makes $IID = IID(p)$

and $NRC = \lfloor NRC(p)/2 \rfloor$ in both packets.

i. **Packet 1.** Contains $d_u \oplus d_p$.

ii. **Packet 2.** Contains only d_u.

(b) However, if $NRC(p) = 1$, node u acts as a node of Type 3, i.e., just a relay.

Note that by taking the floor when updating the NRC value, m is made equivalent to the largest power of 2 that is less than or equal to m. Note also how the resulting tree resembles a binary tree, where each leaf is a combination in our case. However, our tree is restricted in depth, where it can have at most $d = \log_2 m$ levels. Therefore, the maximum number of combinations (or leaves) is $2^d = 2^{\log_2 m} = m$, which is the case only when we have a complete binary tree of d levels. By this we insure that the maximum number of combinations is less than or equal to m and the maximum number of participating sources in less than or equal to $m - 1$. For example, consider the network in Figure 5.2 where m is chosen to be 7. Node a is the initiator node (Type 1 node), by following the rules of operation a will send two packets carrying d_a to b and c with $IID = a$ and $NRC = \lfloor \frac{7}{2} \rfloor = 3$. Node c is a Type 3 node, i.e., it will relay only and will not change the value of the IID or the NRC. Node b is a Type 2 and according to the rules of operation it will produce two packets with $IID = a$ and $NRC = \lfloor \frac{3}{2} \rfloor = 1$. It is obvious that starting with $m = 7$ we can have at most 4 packets produced, which is equivalent to starting with $m = 4$ (i.e., the closest power of 2 less than 7). Therefore, in the remainder of this chapter we will always assume that m is a power of 2.

It is worth mentioning that assigning the NRC value as described in the rules of operation is not optimal in terms of maximizing the number of participating sources in a certain F-process. For example in Figure 5.2, assume that similar to node c node d is also a Type 3 node and the packet forwarded from d will reach the BS on a path composed only of Type 3 nodes. However, assume that node f has a data unit of its own that it wants to combine with the packet received from b. It will not be able to participate since the NRC value in the received packet is 1. Therefore, if node a knew beforehand the source nodes that will relay its packets, it can set the NRC to 1 in the packet sent to c, and to 6 (which is equivalent to 4) in the packet sent to b. Nevertheless, since we assume that a node operates using only its local information,

and the deployed network is dense and uniformly distributed, a realistic assumption that an initiator node can make is that both packets will pass through the same number of sources approximately, and thus divide m by 2. Using simulation, in Section 5.7 we verify that using this assumption results in a performance that is not much worse than the optimal.

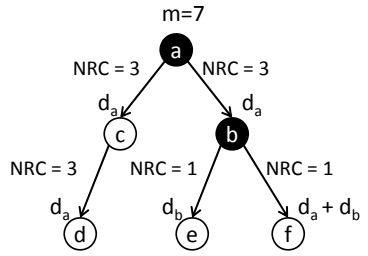


Figure 5.2 NRC update for m=7

5.3 Tolerating Multiple Losses

The operation described in Section 5.2 proactively protects each generated data unit against a single loss. Suppose we want to protect each data unit against e losses. We can do this using the same operation described in Section 5.2 by allowing a Type 1 (Type 2) node, say u, to initiate (participate in) multiple F-processes for the same data unit. Assuming that at most e packets (i.e., combinations) can be lost, d_u must be inserted by u in at least $e+1$ combinations. Since each time u participates in a process two combinations are produced, node u must participate in (or initiate) $\frac{e+1}{2}$ F-processes. Therefore, if e is even (i.e., $e+1$ is odd) either u participates in $\lceil \frac{e+1}{2} \rceil$ F-processes, or in $\lfloor \frac{e+1}{2} \rfloor$ processes and produces a packet containing d_u only with $NRC = 1$ to complete the $e+1$ insertions ($NRC = 1$ to prevent other sources from combining their data with this combination).

Specifically, we assume that each node of Type 1 or Type 2 keeps a set of counters that are initialized to $e+1$, each of which is associated with a certain data unit that is generated by the sensor node itself. Each counter keeps track of the remaining number of participations needed to provide protection against e failures for a certain data unit. A counter is decreased by 2 each time the node participates (or initiates) in a certain F-process until it reaches 0, where no

further participations are needed. An example is shown in Figure5.3, where we want to tolerate 3 packet losses for each of the data units, i.e., $e = 3$. There are three Type 1 nodes (A, B and C), and three Type 2 nodes (D, E and F), each of which has a single data unit to be sent. We only focus on the packets going through nodes D, E and F, where each of these nodes forward traffic from more than one F-process, and we assume that the paths for the remaining packets do no not intersect. For every data unit a counter is initialized to 4, i.e., each Type 1 node can *initiate* 2 F-processes and each Type 2 node can *participate* in 2 F-processes. For example consider node A, the two packets sent from A to nodes D and E represent one process initiated by A, and the remaining two packets represent the other process initiated by A after which the counter will be 0 for d_a. Nodes D and E receive packets from all the three Type 1 nodes, but since they can only participate in 2 processes, they choose to participate in the processes initiated by nodes A and B. Therefore, when nodes D and E receive the packet from C they act as Type 3 nodes and just relay the packet as is. As shown in Figure 5.3, this operation results in a total of 18 packets to protect the data units from 6 sources against 3 losses.

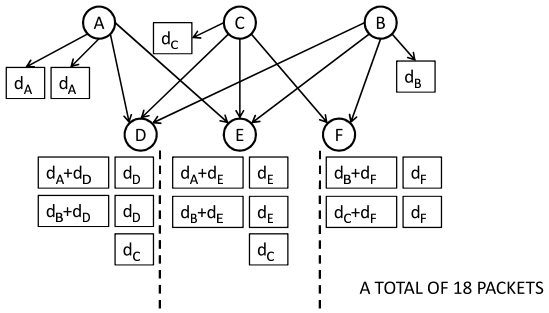


Figure 5.3 Protecting against 3 losses

5.4 Coding/Decoding Issues

The efficiency of network coding is an important issue that rises in most network coding-based applications. In this section we show that the overhead, which results from the need to send the coding vectors along with the combinations to the BS, can be significantly reduced by using relative indexing. Moreover, we introduce the idea of best effort decoding, which allows us to make use of the combinations received at the BS even if more than one packet was lost.

5.4.1 Relative Indexing for Efficient Encoding

For the sink to be able to decode the linear combinations, it needs to have the chosen coefficients or the coding vectors for each one of the combinations. In binary network coding for multicast connections, the length of the coding vector is determined by the minimum max-flow m between the single source and any of the terminals (26), where position i in the vector is reserved for d_i, i.e., if index $i = 1$ then d_i is present in the combination, and if index $i = 0$ then d_i is not. This is possible in multicast connections because all the data units originate from the same source, which means that this single source can assign for each data unit a unique index to identify it with. However, in our case the coding process is distributed, and thus the coding vector should be of size N, where N is the total number of nodes in the network, where each node is a possible source at some time. In a sensor network N can be very large, where it can be in the hundreds or even thousands of nodes. It is obvious that this is a waste of bandwidth since at most $m - 1$ sources can participate in a certain F-process.

We now show how to enable the source nodes in an F-process to use an $(m-1)$-bit coding vector by relatively indexing the sources in that process. To do this, we use the NRC value in addition to a new control bit that we will add to the packet header, which we refer to as I. Assume that both packets generated by an initiator node A will be combined with data units from two other sources B and C. The question is if A is given index 1 in the $(m-1)$-bit coding vector, how can B and C (and any other Type 2 node in the process) choose distinct indices using the values of NRC and control bit I?

In an F-process there is only one node of Type 1, which will be always assigned index 1 (index 0 will not be used). This leaves us with (h-2) indices, which will be recursively divided into halves. Let us consider the pair of Type 2 nodes in the first coding level after A (i.e., B and C), where each of them receives a packet with $NRC = \frac{m}{2}$, but B receives a packet in which $I = 0$ and C receives a packet in which $I = 1$. We give B the first index in the first half, and C the first index in the second half of the remaining h-2 indices. Half of h-2 is $NRC - 1$ (since $m = 2 * NRC$), which means that relative to index 1 the first half begins at the next position **after** *position 1* (i.e., 1+(1)), and the second half begins **after** the next $NRC - 1$ positions

after *position 1* (i.e., 1+($NRC-1$)+1). This is shown in Figure 5.4, where m is 8, the data unit from node B will be given index $1+(1)=2$, and the data unit from node C will be given the index $1+(NRC-1+1)=1+NRC=5$. In the second iteration the process continues but with reference to position $1+(1)$ for the nodes descending from B, and with reference to position $1+(NRC-1)+1$ for the nodes descending from C.

To generalize for other levels, let $X(p)$ be the coding vector of the combination carried in p, and let X_i be an $(m-1)$-bit vector with 1 in the i^{th} position and zeros otherwise. In addition, let $I(p)$ and $NRC(p)$ denote the value of bit I and the NRC field in p respectively. If a node u decides to participate in the F-process of some packet p, it calculates its index according to the following equation:

$$Index(u) = \max_{i:X_i \wedge X(p)=X_i} i + I(p) * (NRC(p)-1) + 1 \tag{5.2}$$

where the values for $I(p)$ and $NRC(p)$ are set according to following operation (we only redefine the operation for Type 1 and 2 nodes):

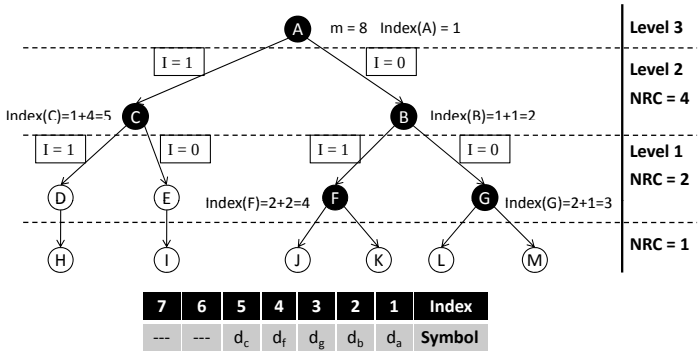


Figure 5.4 Relative indexing, for $m=8$

1. **Type 1.** Assume node u has its own data unit d_u, and has no data units from other sensors to relay. Then node u sends two packets carrying d_u, such that $IID=u$ and $NRC=\lfloor m/2 \rfloor$ in both packets. In addition to the following settings:

   - **Packet 1.** $I(Packet1)=0$, $X(Packet1)=X_1$.

- **Packet 2.** $I(Packet2) = 1$, $X(Packet2) = X_1$.

2. **Type 2.** Assume node u has its own data unit d_u and has received another packet p. The operation of u depends on the NRC value of p as follows:

   (a) If $NRC(p) \geq 2$ node u calculates its index using equation 5.2. Then it produces the following two packets, such that, in both packets $IID = IID(p)$ and $NRC = \lfloor NRC(p)/2 \rfloor$.

      i. **Packet 1.** Contains $d_u \oplus d_p$, and has $I(Packet1) = 0$, and $X(Packet1) = X(p) \oplus X_{Index(u)}$.

      ii. **Packet 2.** Contains only d_u, and has $I(Packet2) = 1$, and $X(Packet2) = X_{Index(u)}$.

   (b) However, if $NRC(p) < 2$ node u acts as a node of Type 3, i.e., just a relay.

For the sake of illustration, we went through one more iteration for the descendents of B, where the NRC is set to 2 in the packets received by nodes F and G, resulting in G taking index 2+(1)=3 and F taking index 2+(2-1 + 1)=4. Lastly, we want to emphasize that relative indexing is valid only for a set of combinations having the same IID.

5.4.2 Best Effort Decoding

For the operation described in Section 5.2, if e combinations were lost from the resulting $1+N_1^j$ combinations in a certain F-process, the sink should not discard the remaining combinations because it may still be able to recover a subset of the encoded symbols. Actually, the remaining combinations received at the BS will still be solvable, if the lost combinations remove $e-1$ data units, i.e., $1+N_1^j-e$ equations remains in $N_1^j-(e-1) = 1+N_1^j-e$ unknowns. From the operation described in Section 5.2 this condition is satisfied for any subtree rooted at a Type 2 node u that combines its data unit with a trivial combination (i.e., a single data unit combination).

Consider a subtree rooted at a Type 2 node u, which encodes its data unit with a single data unit combination containing only d_v. The number of combinations this subtree will produce is

equal to the NRC value in the packet carrying d_v. In addition, the number of new data units that will be added in these combinations by the sources in the subtree (including the root node u) is $NRC - 1$. Therefore, we will have NRC combinations in NRC unknowns, which can be solved. However, if the Type 2 node u combined its data unit with a non-trivial combination, the number of unknowns will be larger than the number of combinations. An example is shown in Figure 5.5, where Type 2 nodes that receive trivial combinations are the heads of all solid links, and Type 2 nodes that receive non-trivial combinations are the heads of all dashed links. For instance the combinations from Subtree 1 are solvable since node d combines its data unit with a trivial combination, while those from Subtree 2 are not.

In reality the BS does not need to search for solvable subtrees to decode them. Rather, it can use the decoding method in algorithm 6 for a set of combinations having the same IID, where RCV is the set that will contain the recovered data units, and Z is a set for temporary use in decoding. This way the BS will recover data units as much as it can, and will stop when no new single data units are found, hence the name best effort decoding.

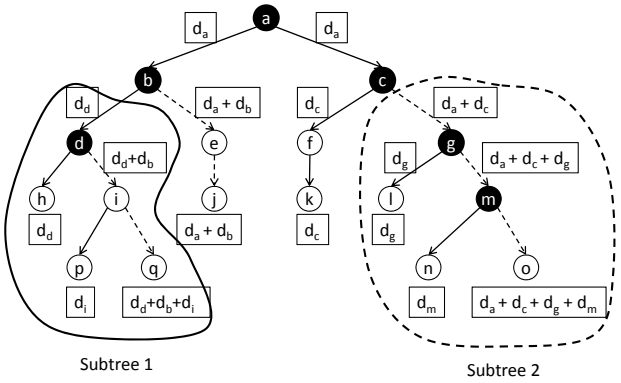


Figure 5.5 Best Effort Decoding

Algorithm 6 Best Effort Decoding

1: $RCV = \phi$
2: Z= All single data unit combinations.
3: **while** $Z \neq \phi$ **do**
4: For each element in Z remove it from all combinations containing it, using bitwise XOR.
5: $RCV = RCV \cup Z$
6: Z= All new single data unit combinations.
7: **end while**

5.5 Routing for maximally disjoint paths

In all of our examples in the previous sections, we assumed that the data units or combinations will select routes to the BS in a way that constructs a binary tree. If we are considering packet loss only, then we do not need to restrict the routing of the combinations to the BS to be a tree. However, if we want to use our coding scheme to tolerate actual node or link failures, then we need a routing protocol that selects for each combination a path that is maximally disjoint from the paths of all other data units or combinations. Note that if all the paths are totally disjoint, then the routes from the sources in the different coding levels will form a tree. Therefore, to be able to tolerate node or link failures, we present a simple routing protocol that guarantees maximally disjoint paths in this section.

For our routing protocol to work we need the underlying WSN to be organized into levels or rings around the BS, as assumed at the beginning of the chapter. Initially, a node (say v) has its level l_v set to a very large value. Arranging the network into levels can be done through a simple process initiated by the BS. The BS starts by broadcasting a control packet containing a field called *hop_count*, which is initialized to 1. After that, if a node v receives this control packet with $hop\_count \geq l_v$, it discards the packet. Otherwise, it makes $l_v = hop\_count$, increases the *hop_count* by 1 and rebroadcasts the control packet.

WSNs are usually densely deployed. Therefore, during the process of organizing the network into levels, a node, v, will most likely receive more than one packet with the same minimum *hop_count* from nodes in $l_v - 1$. Each node stores the IDs of these neighbors in a list called *next_hop_list*. If a node needs to forward a data packet to the BS, it needs to only send it to one of the nodes stored in the *next_hop_list*.

We now define the operation for Type 3 nodes to produce maximally disjoint paths (the operation for Type 2 nodes is similar, but takes into account two packets instead of one). To establish maximally node-disjoint paths from the multiple sources in the same F-process, a Type 3 node must not forward more than one packet from the same F-process (i.e., with the same IID). Note that to establish maximally edge-disjoint paths, a Type 3 node can forward multiple packets with the same IID, but not to the same next hop (unless it is forced to

do so). To do this, each node needs to know for which F-processes it had forwarded packets before. Therefore, each node stores the IID of each packet it forwards in a list called the IID_list. If node v is forwarding a packet p with $IID(p)$, it must select a neighbor in the $next\_hop\_list$ that does not have $IID(p)$ in its IID_list. This can be accomplished during the RTS/CTS negotiation, where the IID (say $IID(p)$) of packet p (which needs to be transmitted) is piggybacked in the RTS message, and a next-hop node in $l_v - 1$ can reply with a CTS if it does not have $IID(p)$ in its IID_list. Node v may not be able to find a neighboring node in $l_v - 1$ that does not have $IID(p)$ in its list. Therefore, to solve this problem, we add a new field to the RTS message, which is basically a flag (bit) called *"Force forward"* or simply FF. A node can force forward a packet, if it did not receive a CTS, by setting $FF = 1$ in the RTS. If a neighboring node in $l_v - 1$ heres an RTS with $FF = 1$ it replies with a CTS regardless if it has $IID(p)$ or not. The operation of a node upon receiving a packet is described in Algorithm 7, and its operation when sending a packet is described in Algorithm 8. Although Algorithms 7 and 8 do not show it, but we assume that some back off mechanism is used before a packet is sent.

Algorithm 7 Operation of node v: Receiving packet p

```
1: if (p is a control packet) then
2:    if (hop_count > l_v) then
3:       Discard p
4:    else if (hop_count < l_v) then
5:       l_v = hop_count
6:       Clear next_hop_list
7:       Rebroadcast p
8:    else
9:       //hop_count = l_v
10:      Add sender of p to next_hop_list
11:      Rebroadcast p
12:   end if
13: else if (p is RTS received from level l_v + 1) then
14:   if (if IID(p) ∉ IID_list) then
15:      Send CTS
16:   else if (FF(RTS) == 1) then
17:      Send CTS
18:   end if
19: else
20:   //p is a data packet
21:   Store IID(p) in IID_list
22:   Send p downstream
23: end if
```

Algorithm 8 Operation of node v: Sending packet p

```
1: if (p is a control packet) then
2:    Broadcast p
3: else
4:    //p is a data packet
5:    Piggyback IID(p) on RTS
6:    if (CTS is received from node u before timeout) then
7:       Send p to node u
8:    else
9:       FF(RTS) = 1
10:      Wait for CTS
11:      Upon receiving CTS from node u, send p to u
12:   end if
13: end if
```

It is easy to see that by using the described routing protocol, all paths from all sources will be node-disjoint unless some node force forwards a packet. Also, the paths will be link-disjoint unless a node forwards two packet to the same next hop (one packet is normally forwarded, but the other is force forwarded to the same next hop). By using such a simple routing protocol, we can say that the F-process can tolerate at most one node (or link) failure if the paths from all sources are node (or link) disjoint. An example is shown in Figure 5.6, where an F-process is initiated by node A, and node B is only downstream Type 2 node. In the example, node C force forwards the packet carrying data unit a because there are only two nodes in $l_C - 1$. After that, no force forwarding occurs, which produces edge-disjoint paths. Note that if another neighbor to node C is found in $l_C - 1$, then the paths would be node-disjoint. However, note that if there were only two nodes in $l_E - 1$, then E will force forward one of the data units a or b, and the resulting paths will not be even edge-disjoint. It should be noted that if the paths are totally disjoint, then the failure of a node can cause at most one combination to be lost even if it is a Type 2 node. Let v be a Type 2 node that receives packet p from some F-process, then if v fails before sending its two packets containing $d_v \oplus d_p$ and d_v, the failure of v will only cause d_p to be lost.

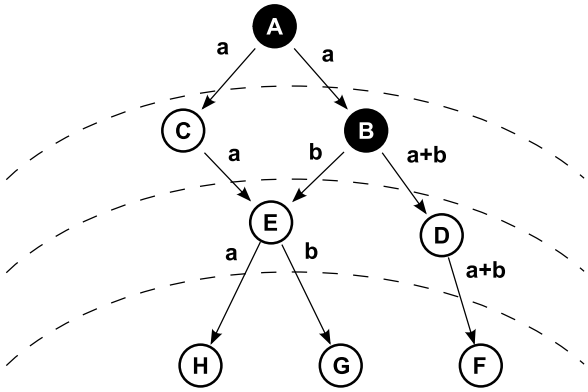


Figure 5.6 The figure shows how the combinations created in an F-process initiated by node A are forwarded. Node C cannot find a downstream neighbor that did not forward a packet with IID = A. Therefore, it force forwards data unit a to node E, which results in edge-disjoint paths. Note that if another neighbor to node C is found the paths would be node-disjoint. Note also that if there were only two nodes in $l_E - 1$, then E will force forward one of the data units (either a or b), and the resulting paths will be neither link nor node-disjoint

5.6 Selecting parameter m

The min-cut is not the only factor that limits the number of combinations m. Any network link may be operational with a certain probability. As the value of m gets larger more sources will be able to participate in an F-process, which in turn increases the number of used network links, and thus increases the chance of having multiple link failures that can affect the ability of the sink to successfully decode the received combinations in a certain F-process. In this section we study the relationship between m and the probability of successful recovery $P(rcv)$. We assume that all network links have the same success probability, which we denote by q. In the following discussion, we focus on a single F-process and we show how $P(rcv)$ changes with respect to the value of m.

Since we need sources to participate in an F-process to further reduce the redundancy overhead, we need to know how the participation of these sources will affect $P(rcv)$. The

probability of successful recovery depends on the number of used links, which in turn depends on three factors: 1) L, the level of the initiator node, 2) $m-1$, the number of sources that will participate in the F-process, and 3) the way the sources are distributed in the levels below the initiator node, or equivalently the resulting topology of the tree. The probability of successful recovery given some known topology, Tp_1, with k_1 links, is equal to the probability that at most one loss occurs in Tp_1, i.e.,:

$$\begin{aligned} P(rcv|Tp_1) &= P(\text{no loss in } Tp_1) + P(1 \text{ loss in } Tp_1) \\ &= q^{k_1} + k_1(1-q)q^{k_1-1} \end{aligned}$$

Therefore, we can compute $P(rcv)$ as follows:

$$P(rcv) = \sum_{\forall Tp_i} P(rcv|Tp_i)P(Tp_i)$$

Since we assume that all network nodes generate data units at the same rate, i.e., the probability of being a source is the same for all nodes, then all possible topologies can occur with the same probability. Therefore, we can write the probability of successful recovery as a function of L and $m-1$ as follows:

$$\begin{aligned} P(rcv) = P_{rcv}(L, m-1) &= \frac{\sum_{\forall Tp_i} P(rcv|Tp_i)}{\#\text{possible topologies}} \\ &= \frac{\sum_{\forall Tp_i} (P(\text{no loss in } Tp_i) + P(1 \text{ loss in } Tp_i))}{\#\text{possible topologies}} \\ &= \frac{P_0(L, m-1) + P_1(L, m-1)}{\#\text{possible topologies}} \end{aligned}$$

$P_0(L, m-1)$ is the sum of probabilities of no loss in all the possible topologies starting

at level L with $m-1$ sources, and $P_1(L, m-1)$ is the sum of probabilities of exactly 1 loss in all the possible topologies starting at level L with $m-1$ sources. Following the rules of operation, the initiator node will send two packets containing copies of its data unit to the BS. As these packets are forwarded, a Type 2 node may participate and code its data with the data unit received from one of these packets, which also produces two packets according to the rules of operation that may also trigger other downstream Type 2 nodes to participate. For the purpose of computing $P(rcv)$, we can ignore the contents of the packets forwarded from Type 1 or Type 2 nodes, and just focus on the number of transmissions or packets produced, we can view an F-process (initiated by a Type 1 node) as a recursive process that repeats itself (in some lower level, and with a fewer number of sources) on every downstream source (Type 2 node). This allows us to calculate $P_0(L, m-1)$ and $P_1(L, m-1)$ using recursive formulas, but first we need the following definitions:

- let $\pi(\lambda, \alpha)$ be the probability that there is no loss in the branch starting at level λ, with α sources downstream.
- let $\bar{\pi}(\lambda, \alpha)$ be the probability that there is a loss in the branch starting at level λ, with α sources downstream.
- let $\mu(\lambda, \alpha)$ be the probability that a loss will occur downstream, given that we are starting at level λ with α sources and no loss has occurred yet.
- let $\nu(\lambda, \alpha)$ be the probability that no loss will occur downstream, given that we are starting at level λ with α sources and a loss has already occurred.
- let $\rho(\lambda, \alpha)$ be the probability that no loss will occur downstream, and the root is not necessarily a source. This is similar to $P_0(\lambda, \alpha)$, but branching may not happen at level λ.

Using these definitions we can write P_0 and P_1 as follows:

$$P_0(L,m-1) = \sum_{\alpha_1=0}^{\lfloor \frac{m-1}{2} \rfloor} \sum_{\forall\{\alpha_2:\alpha_1+\alpha_2=m-2,\ 0\leq\alpha_2\leq\lfloor \frac{m-1}{2} \rfloor\}} \pi(\lambda,\alpha_1)\pi(\lambda,\alpha_2) \tag{5.3}$$

$$\begin{aligned} P_1(L,m-1) = &\sum_{\alpha_1=0}^{\lfloor \frac{m-1}{2} \rfloor} \sum_{\forall\{\alpha_2:\alpha_1+\alpha_2=m-2,\ 0\leq\alpha_2\leq\lfloor \frac{m-1}{2} \rfloor\}} \pi(\lambda,\alpha_1)\bar{\pi}(\lambda,\alpha_2) \\ &+ \sum_{\alpha_1=0}^{\lfloor \frac{m-1}{2} \rfloor} \sum_{\forall\{\alpha_2:\alpha_1+\alpha_2=m-2,\ 0\leq\alpha_2\leq\lfloor \frac{m-1}{2} \rfloor\}} \bar{\pi}(\lambda,\alpha_1)\pi(\lambda,\alpha_2) \end{aligned} \tag{5.4}$$

where,

$$\pi(\lambda,\alpha) = q.\rho(\lambda-1,\alpha) \tag{5.5}$$

with $\pi(1,0) = q$, and $\pi(1,\alpha \geq 1) = 0$ because there are no sources below level 1.

$$\bar{\pi}(\lambda,\alpha) = q.\mu(\lambda-1,\alpha) + (1-q).\nu(\lambda-1,\alpha) \tag{5.6}$$

with $\bar{\pi}(1,0) = 1-q$, and $\bar{\pi}(1,\alpha \geq 1) = 0$.

$$\mu(\lambda,\alpha) = q.\mu(\lambda-1,\alpha) + (1-q).\nu(\lambda-1,\alpha) + P_1(\lambda,\alpha) \tag{5.7}$$

with $\mu(1,0) = 1-q$, $\mu(1,1) = 2q(1-q)$, and $\mu(1,\alpha > 1) = 0$.

$$\nu(\lambda,\alpha) = \nu(\lambda-1,\alpha) + P_0(\lambda,\alpha) \tag{5.8}$$

with $\nu(1,0) = q$, $\nu(1,1) = q^2$, and $\nu(1,\alpha > 1) = 0$.

$$\rho(\lambda,\alpha) = P_0(\lambda,\alpha) + q.\rho(\lambda-1,\alpha) \tag{5.9}$$

with $\rho(1,0) = q$, $\rho(1,1) = q^2$, and $\rho(1,\alpha > 1) = 0$.

Finally, we need to compute the total possible number of topologies to be able to calculate $P_{rcv}(L, m-1)$. To do this we can use $P_0(L, m-1)$. Since P_0 computes the sum of the probability of no loss in all possible topologies setting $q = 1$ will make the probability of no loss equals 1 for each topology, and P_0 will result in counting the total number of possible topologies. That is, $P_{rcv}(L, m-1)$ can be calculated as follows:

$$P_{rcv}(L, m-1) = \frac{P_0(L, m-1) + P_1(L, m-1)}{P_0(L, m-1)|_{q=1}} \tag{5.10}$$

The probability of successful recovery is evaluated in the next section.

5.7 Performance Evaluation

In this section we evaluate the probability of successful recovery, $P(rcv)$, based on the formula derived in the previous section. We also present simulation results from our implementation in TOSSIM, to illustrate the benefits of our scheme in reducing the protection overhead.

5.7.1 Evaluating $P(rcv)$

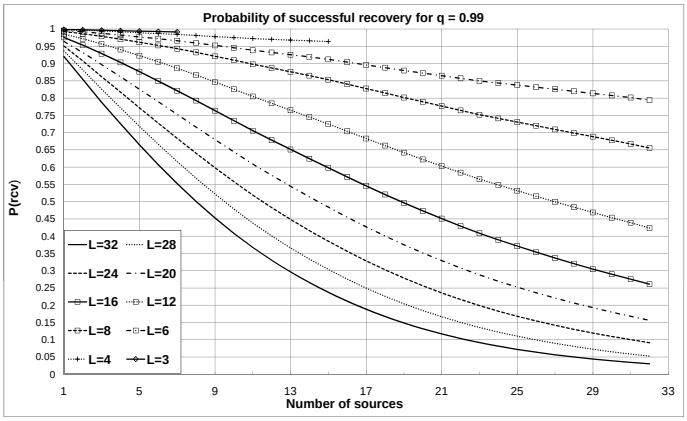
We plotted $P(rcv)$ against the total number of sources in an F-process starting at different levels, for $q = 0.99$, $q = 0.999$, and $q = 0.9999$. The results are shown in Figure 5.7. As

expected, for some level L, increasing the number of participating sources in an F-process will reduce $P(rcv)$ because this increases the total number of used links. Therefore, depending on the desired probability of successful recovery, nodes in levels that are farther from the BS may want to just use duplication, or equivalently let $m = 2$ to prevent downstream Type 2 nodes from participating. Also, for a fixed number of sources, the figure shows that the probability of successful recovery increases as the initiator node gets closer to the BS (i.e., as L decreases).

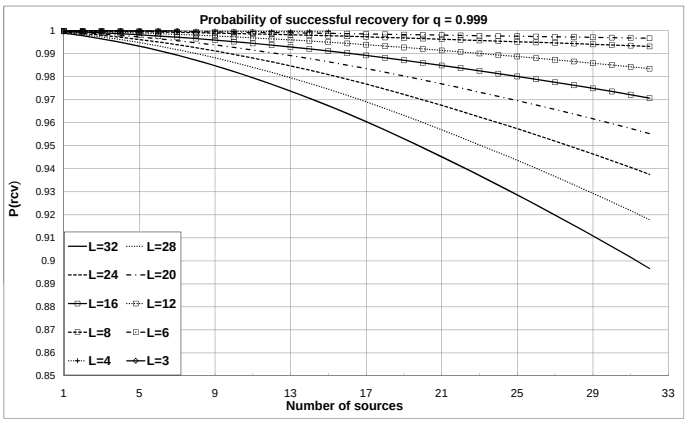
Note that the total number of combinations, m, produced by an F-process that has L levels is limited by 2^L, and that the total number of source nodes is limited by $2^L - 1$. We will see from the simulation results in the next section that we do not actually need a large number of Type 2 nodes to participate in an F-process to significantly reduce the duplication overhead. This is due to the fact that the reduction in the duplication overhead becomes negligible after the first few sources participate. Finally, it should be noted that all these calculations for $P(rcv)$ represent lower bounds, and are only valid if we are considering the recovery of the data units from all the sources participating in an F-process. This is because as we have shown in Section 5.4.2, our coding scheme allows the partial recovery of a subset of the coded data units if they are from a subtree rooted at a Type 2 node that receives a native data unit. Therefore, the chances for recovery are better than what Figure 5.7 implies.

5.7.2 Evaluating the produced overhead

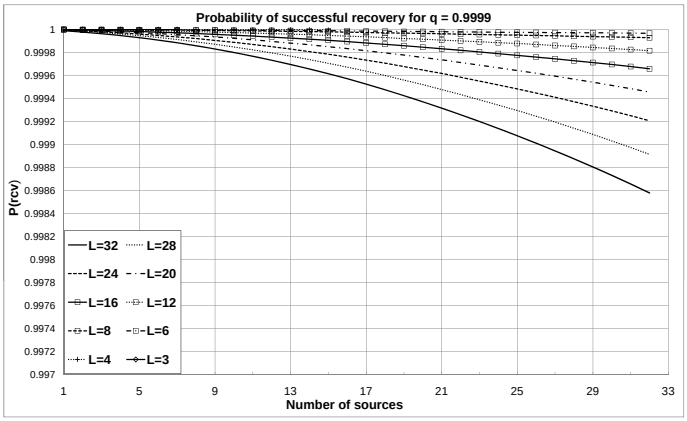
We evaluated the performance of our scheme in terms of produced overhead, using simulation on TOSSIM. The effects of the following three parameters were studied in our experiments: 1) S, which represents the number of packets generated by each sensor node, 2) the number of nodes in the network, denoted by N, and 3) m, the maximum number of produced combinations in an F-process, which controls the degree of coding. As a reference, we compared our results to a theoretical lower bound, which represents the best way in which packets can be combined. The best case occurs when every $m - 1$ nodes belong to a certain F-process in which all of them are either Type 1 or Type 2 nodes. That is, we will have $\frac{N}{m-1}$ groups of nodes each of which produces a total of mS packets. This gives the following lower bound on



(a) $q = 0.99$



(b) $q = 0.999$



(c) $q = 0.9999$

Figure 5.7 The figures show how the probability of recovery $P(rcv)$ changes with respect to the initiator node level (L) and the number of participating sources.

the total number of packets produced:

$$(\frac{m}{m-1})N * S$$

We experimented with three scenarios, where in all of these scenarios m took the values {2, 4, 8, 16}. Note that when $m = 2$ no coding will take place and our scheme will be equivalent to traditional duplication based redundancy. The first scenario is a 7x7 grid network where each node produced 20 packets, the result is shown in Figure 5.8, where at $m = 16$ the duplication overhead is reduced by 42%. The second is also a 7x7 grid, but each node produced 50 packets to see how increasing the load affects the performance. Figure 5.9 shows that by increasing the number generated packets our approach performs better and decreases the duplication overhead by 50% at $m = 16$. Finally, we increased the size to a 10x10 grid network and fixed the number of produced packets to 50, to see how the performance is affected by increasing the network size in addition to the number of packets. Again, the results assure that the performance gets better with increasing the network size and generated packets, where at $m = 16$ the duplication overhead is reduced by 72%. It is clear that the third scenario is the closest to the lower bound, since the chances for combining packets are enhanced as the network and/or number of generated packets grows larger. Although these scenarios are relatively small in size, they clearly illustrate that our coding-based approach is scalable and performs better as the network size increases.

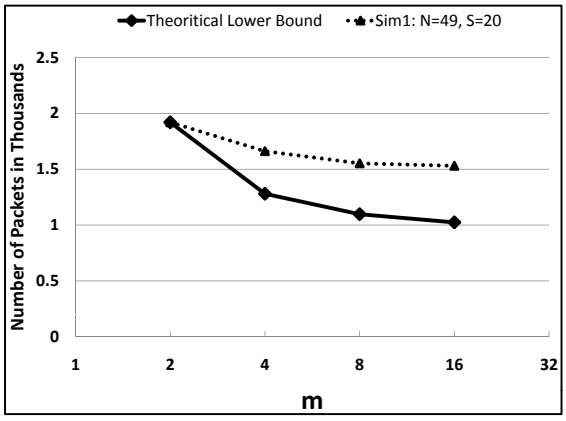


Figure 5.8 Simulation results for N=49 and S=20

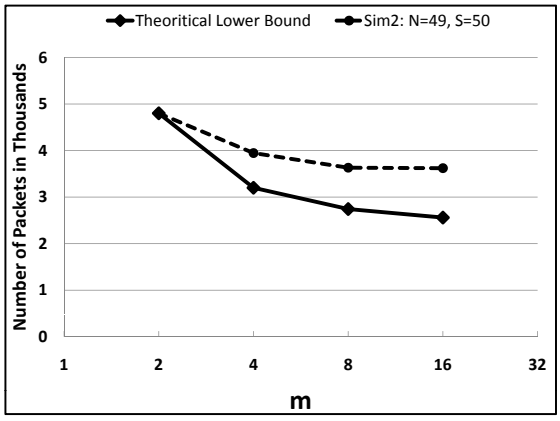


Figure 5.9 Simulation results for N=49 and S=50

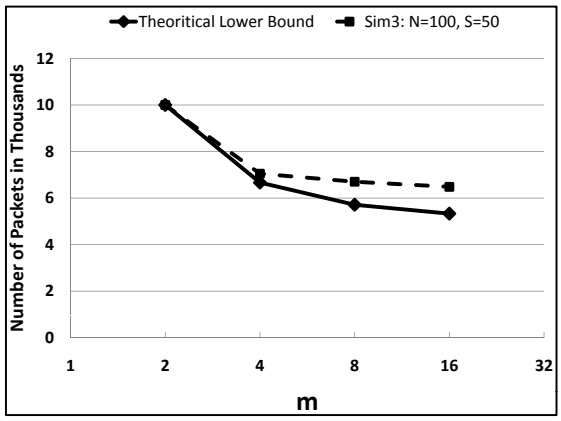


Figure 5.10 Simulation results for N=100 and S=50

To see how the probability of successful recovery ($P(rcv)$) affects the overhead, we modified the results in Figure 5.10 to reflect the change in $P(rcv)$. We divided the number of packets for each point in Figure 5.10 by the corresponding $P(rcv)$ (at $m-1$) from Figure 5.7(a). The result is shown in Figure 5.11. It is clear from the figure that network coding might not always be better than duplication especially if an F-process is initiated at a level that is far from the BS. The figure also shows that a good reduction in the overhead can be achieved by just letting a small number of sources to participate in an F-process, e.g., 3 (at m = 4).

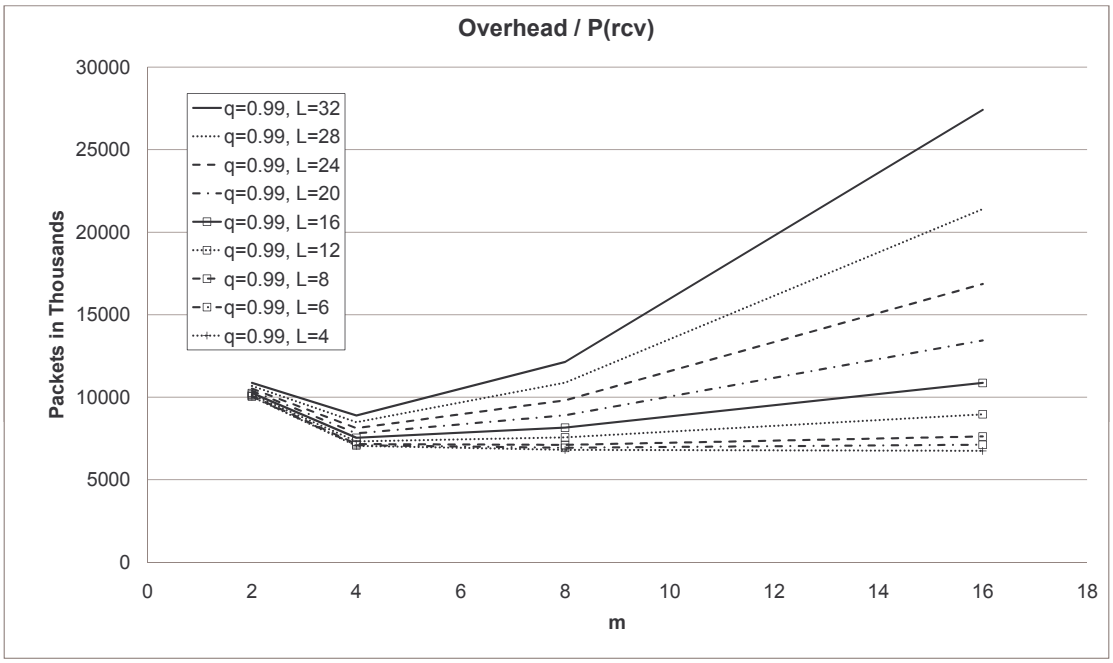


Figure 5.11 The figure shows how the produced overhead changes with respect to the initiator node level (L), and the number of participating sources.

5.8 Summary

Deterministic binary network coding was utilized in a distributed manner to add lightweight redundancy to the information flow from the sensor nodes to the base station. Relative indexing was introduced to enable the use of an $(m-1)$-bit coding vector instead of an N-bit coding vector, where N is the number of sensor node (which can be in thousands), and $m << N$. In addition, we presented best effort decoding that allows us to make use of the combinations received at the BS even if more than one packet was lost. We enabled our scheme to tolerate link and node failures by presenting a simple routing protocol that guarantees maximally disjoint paths for the produced combinations in an F-process. We also studied the relationship between the probability of successful recovery, $P(rcv)$, and the number of sources participating in a certain F-process that is initiated from some level L. Finally, simulation results confirmed our theory and proved that our scheme is highly scalable, where it performs better as the network size and/or the number of sources increases.

CHAPTER 6. Max-flow protection using network coding

6.1 Introduction

As has been shown in previous chapters, network coding-based protection and traditional protection schemes, provide end-to-end protection of the whole S-T paths used to forward useful data from S to T. In these approaches, the more we enhance the S-T flow survivability, the more we reduce the useful S-T information rate. This is because such approaches treat all network links equally, i.e., bottleneck links (that belong to the min-cut) as well as non-bottlenecks are used to forward redundant data units or combinations. Usually, most of the links in a network are not bottleneck links, which means that link failures are more likely to affect non-bottleneck links than links in the min-cut. Therefore, we can enhance the survivability of the S-T information flow without reducing the useful S-T rate below the max-flow, if we provide protection to the non-bottleneck links only. We call this kind of protection *Max-flow protection* because the max-flow can still be achieved under these conditions as long as no link in the min-cut fails. In this chapter, we focus on the problem of max-flow protection. We introduce the idea of extra source or destination connectivity with respect to a certain S-T max-flow, and then we study two problems: namely, pre-cut protection and post-cut protection. Although our objective in both problems is the same, where we aim to maximize the number of protected paths, our analysis shows that the nature of these two problems are very different, and that the pre-cut protection problem is much harder. Specifically, we prove the hardness of the pre-cut protection problem, formulate it as an integer linear program (ILP), and propose a heuristic approach to solve it. Simulations show that the performance of the heuristic is acceptable even on relatively large networks. In the post-cut problem we show that all the data units, forwarded by the min-cut edges not incident to T, can be post-cut-protected.

The rest of this chapter is organized as follows. Section 6.2 presents the terminology and definitions that will be used throughout the chapter. The problems of pre-cut and post-cut protection are presented in Section 6.3. In Section 6.4 we study the pre-cut protection problem and prove its hardness. The problem of pre-cut protection is formulated as an Integer Linear Program (ILP) in Section 6.5. A 3-phase heuristic approach to solve the pre-cut protection problem is described in Section 6.6. Section 6.7 discusses the post-cut protection problem. Finally, the chapter is summarized in Section 6.8.

6.2 Preliminaries

We represent a network by a directed acyclic graph G=(V,E), where V is the set of network nodes and E is the set of available links, where each link is assumed to have unit capacity. The network has a source node (S) that wants to send data to a destination (T), where the S-T max-flow is assumed to be h. We assume that a multipath routing protocol is used, and that the source is fully utilizing the available connectivity by sending h data units to the destination simultaneously. To simplify the analysis, we assume that the network has a single cut. In the rest of this section we define the meaning of extra connectivity with respect to the S-T max-flow. After that we discuss some of the properties of nodes with extra connectivity.

6.2.1 Terminology

Let $f^{(A)}(B)$ denote the max-flow from the nodes in set A to the nodes in set B on a directed graph, which can be calculated by computing the max-flow between a virtual source/sink pair, such that the virtual source is connected to the nodes in A with infinite capacity edges and the virtual sink is connected to the nodes in B with infinite capacity edges also. Let $h = f^S(T)$ be the S-T max-flow. We define the following:

1. A node with Extra Source Connectivity (wESC) is a node, u, that satisfies the following conditions:

   - $f^S(u,T) > h$, and $f^{(S,u)}(T) = h$.

2. A node with Extra Destination Connectivity (wEDC) is a node, v, that satisfies the following conditions:

   - $f^S(v,T) = h$, and $f^{(S,v)}(T) > h$.

3. A node with No Extra Connectivity (wNEC) has:

   - $f^S(v,T) = f^{(S,v)}(T) = h$.

Of course, a node with both extra source and extra destination connectivity cannot exist, because this contradicts the assumption that the max-flow equals h. Consider the graph G in Figure 6.3. The S-T max-flow in G is 4, which implies that four data units can be forwarded from S to T on four link-disjoint paths. Assume we found the following paths, $P_1 = \{S \rightarrow A \rightarrow E \rightarrow J \rightarrow T\}$ that forwards data unit w, $P_2 = \{S \rightarrow B \rightarrow F \rightarrow G \rightarrow T\}$ that forwards data unit x, $P_3 = \{S \rightarrow F \rightarrow H \rightarrow T\}$ that forwards data unit y, and $P_4 = \{S \rightarrow D \rightarrow I \rightarrow T\}$ that forwards data unit z. Each path P_i contains a cutting edge C_i, which , if deleted, will result in reducing the max-flow by exactly 1 unit of flow because path P_i will be disconnected and cannot be reestablished in any way. In our example, P_1 contains $C_1 = \{(J,T)\}$, P_2 contains $C_2 = \{(G,T)\}$, P_3 contains $C_3 = \{(F,H)\}$, and P_4 contains $C_4 = \{(I,T)\}$.

6.2.2 Properties of nodes wESC/wEDC

Consider a path that contains a node, u, wESC and a node, v, wEDC. Note that node u must be closer (in number of hops on the path) to the source than v, otherwise the max-flow assumption will be contradicted, as shown in Figure 6.1.

In general, removing the min-cut edges (i.e., the edges in $\cup_{i=1}^{h} C_i$) partitions the network into two partitions A and A', such that $S \in A$ and $T \in A'$. Note that, after deleting the min-cut edges, each of the partitions A and A' is a connected component (at least weakly), and that partition A contains nodes wESC, but partition A' contains nodes wEDC.

Lemma 4. *Any node $u \in A, u \neq S$ is a node wESC.*

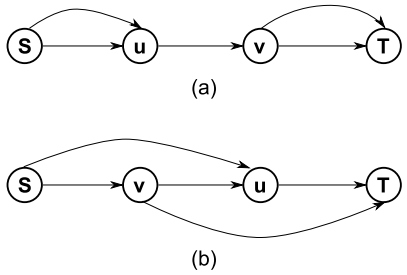


Figure 6.1 In (a) we can say that the S-T max-flow is 1 and that u is a node wESC and v is a node wEDC. However, this is not true in (b), because there are two edge-disjoint paths $\{S \rightarrow u \rightarrow T\}$ and $\{S \rightarrow v \rightarrow T\}$, i.e., u is not a node wESC and v is not a node wEDC

Proof. We prove this by contradiction. Let $u \in A, u \neq S$, but u is not a node wESC. Then, $f^S(u,T) = h$, which means that node u cannot receive additional flow from S if the S-T max-flow is established. This implies that either node u is behind the min-cut (i.e., $u \in A'$), which contradicts the starting assumptions, or that there is another min-cut between S and u, which contradicts the single min-cut assumption. □

In a similar fashion, we can prove the following for any node $v \in A', v \neq T$.

Lemma 5. *Any node $v \in A', v \neq T$ is a node wEDC.*

In our following discussion we refer to A as the *pre-cut* portion of the network, and to A' as the *post-cut* portion of the network. Figure 6.2 summarizes the previous discussion.

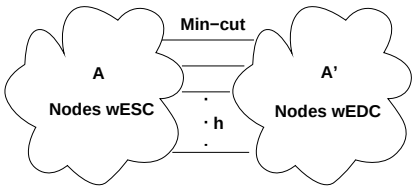


Figure 6.2 Nodes wESC, wNEC and wEDC with respect to min-cuts

6.3 Problem description

The cutting-edges, cannot be protected unless we trade bandwidth for survivability (i.e., unless we use an S-T path to carry redundant information to the destination), which reduces the useful S-T information rate. This tradeoff not only protects the cutting-edges, but also protects any edge carrying data in the network. However, the non-cutting-edges (or a subset of them) can be protected without reducing the S-T information rate, if the graph contains nodes wESC and/or wEDC. For example, nodes E, F, I and J in Figure 6.3 are nodes wESC, and node H is a node wEDC. There are four possible ways to utilize the extra source connectivity in Figure 6.3; 1) protect data units x and y by sending $x + y$ to F through C, 2) protect w by sending a duplicate to E through C and F, 3) protect w by sending a duplicate to J through C, F, E and G 4) protect z by sending a duplicate to I through C and F. The first option is better than the other three since sending $x + y$ to F enhances the chances of two data units (x and y) to reach T, compared to duplicating w or z alone, which protects a single data unit only. Figure 6.4 shows the first option, and it also shows how to utilize the extra destination connectivity from node H, where H sends a duplicate of y to T through node K.

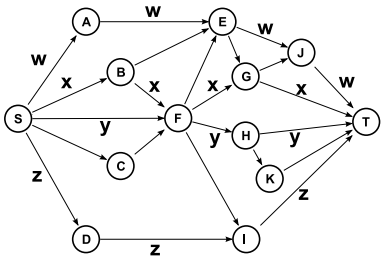


Figure 6.3 Graph G with S-T max-flow = 4

In this chapter, we propose a different way to handle the "survivability vs. bandwidth" trade-off. We propose a new approach to provide protection to the S-T information flow without reducing the useful S-T data rate. Basically, we avoid protecting the bottlenecks in the network (the min-cut links), and we try to efficiently utilize (by using network coding if possible) the available network connectivity before and/or after the bottleneck to provide protection to the non-min-cut links in the graph. We divide the problem into two sub-problems

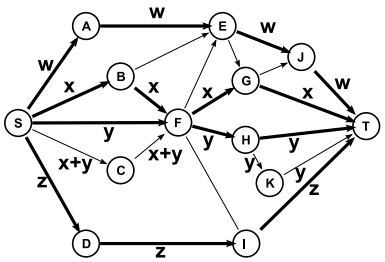


Figure 6.4 Utilizing extra connectivity

as follows:

1. Pre-cut protection: Our objective is to maximize the number of pre-cut-protected S-T paths. We show that this problem is NP-hard, and we provide a heuristic to solve it. To evaluate our heuristic we compare its performance to an ILP.

2. Post-cut protection: Similar to the previous objective, we aim to maximize the number of post-cut-protected S-T paths. Let e_i be the closest cutting edge to the destination T on path P_i. We show that all the paths that do not have T as the head node of e_i , where $1 \leq i \leq h$, can be post-cut-protected together against at least one failure.

6.4 Pre-Cut: Nodes with Extra Source Connectivity

As discussed in Section 6.2, all nodes wESC are located in the pre-cut portion of the network. Assume that the set $\mathcal{X}$ contains all the nodes wESC, $\mathcal{X} = A \backslash S = \{u_1, u_2, \ldots, u_{|\mathcal{X}|}\}$. Then, the following is true:

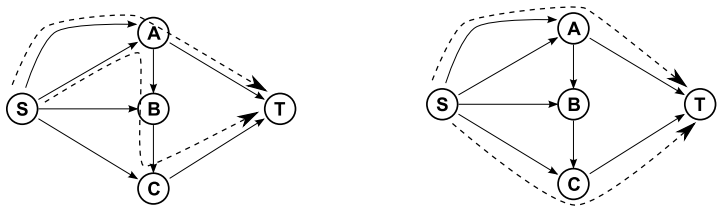
$$(\sum_{i=1}^{|\mathcal{X}|} f^S(u_i, T)) - |\mathcal{X}| f^S(T) \geq f^S(u_1, u_2, \ldots, u_{|\mathcal{X}|}, T) - f^S(T) \tag{6.1}$$

This is because the extra source connectivity may be shared between the nodes in $\mathcal{X}$. Therefore, the right hand side of the inequality is what really determines the available extra source connectivity (ESC). This implies that not all nodes wESC in $\mathcal{X}$ can receive redundant flows from S to be used to protect the S-T max-flow, and thus, a subset $X \subseteq \mathcal{X}$ should be intelligently selected to receive the available extra source flow and utilize it in the best way

possible. Note that the number of nodes in X cannot exceed the extra available connectivity, i.e.:

$$ESC = f^S(u_1, u_2, \ldots, u_{|\mathcal{X}|}, T) - f^S(T) \geq |X| \tag{6.2}$$

The selection of X depends on how the S-T max-flow is routed on the graph. Consider the graph in Figures 6.5(a) and 6.5(b), the S-T max-flow in this network is 2, and there is only one S-T min-cut in the graph, which contains the edges (A,T) and (C,T). Nodes A, B and C are nodes wESC, and the total available extra source connectivity equals $f^S(A, B, C, T) - f^S(T) = 4 - 2 = 2$. Assume that the max-flow is routed as shown in Figure 6.5(a) (the dashed lines), in this case $X_1 = \{B, C\}$ since the extra source connectivity is consumed by B and C. Moreover, note that only the path forwarding **b** can be pre-cut-protected by sending copies of **b** on (S, B) and (S, C). Now consider the routing shown in Figure 6.5(b), in this case $X_2 = \{A, C\}$. Unlike the previous case, both paths can be pre-cut-protected by sending a second copy of **a** to A, and a second copy of **b** to C through B. Obviously, the second routing option is better since it allows the protection of both paths (equivalently both data units), in this sense we say X_2 is better than X_1.



(a) (b)

Figure 6.5 Routing the max-flow is what determines X. In (a) $X = \{B, C\}$, and one path is protected. In (b) $X = \{A, C\}$, and both paths are protected.

It was shown in the previous example that routing the max-flow and selecting X are inseparable problems, and that routing the S-T max-flow corresponds to selecting X. Let us define the extra source connectivity to a node u with respect to the routing of the S-T

max-flow in the network as:

$$EC(u) = f^S(u,T) - f^S(T)$$

We say that an S-T path is pre-cut-protected if a segment of this path in the pre-cut portion of the network is protected. That is, a path is pre-cut-protected if it contains a node wESC with respect to the routing of the S-T max-flow. Therefore, maximizing the number of pre-cut-protected paths means maximizing the number of paths containing nodes wESC.

For large networks, trying-out all possible routing choices to find the best one that will maximize the number of paths containing nodes wESC is computationally expensive. The following theorem proves that this problem is in fact an NP-hard problem.

Theorem 7. *Routing the S-T max-flow to maximize the number of S-T paths containing nodes wESC is NP-hard.*

Proof. To prove this theorem, we reduce the Maximum Coverage problem with Group budget constraints (MCG) (28) to our problem. For the reader's convenience we restate the MCG problem:

Given: A collection of sets $C = \{C_1, C_2, ..., C_m\}$ that are not necessarily disjoint, where each set is a subset of a given ground set H. In addition, C is partitioned into disjoint groups $\{G_1, G_2, ..., G_n\}$, where G_j consists of a group of sets in C.

Question: Can one select k sets from C to maximize the cardinality of their union, such that at most one set from each group is selected?

We reduce the covering problem to our routing problem by transforming any instance of MCG to a graph as follows:

- For each element in H we have two corresponding nodes e_i and e'_i, $1 \leq i \leq |H|$.
- For each set, C_i, we have a corresponding node with the same label C_i.
- For each group, G_i, we have a corresponding node with the same label G_i.
- Finally, we add dummy source and destination nodes S and T.

To complete the transformation we need to add edges to our graph. The nodes above are connected as follows:

- Connect S with every G_j using unit capacity links (S, G_j), $\forall j$.
- Connect S with every e_i using c_i links of unit capacity, where c_i is equal to the number of subsets containing e_i, i.e., if e_i belongs to 4 subsets then there are 4 unit capacity links between S and e_i.
- Connect every e'_i to T using c_i unit capacity links also.
- Connect a node C_j with the two nodes corresponding to element e_i using unit capacity links (e_i, C_j) and (C_j, e'_i), if $e_i \in C_j$.
- Connect a node G_j with a node C_i using a unit capacity link (G_j, C_i),if $C_i \in G_j$.

The resulting graph will look similar to that in Figure 6.6. Note that each (S, e_i) link corresponds to c_i links, we did not draw all the links to make the figure more readable. Note that the S-T max-flow in this figure equals the in-degree of T, which equals $\sum_{i=1}^{|H|} c_i$. Note also that all the nodes in the same group share the extra source connectivity going through the nodes representing the groups. It remains to show that finding the S-T max-flow that maximizes the number of paths containing nodes wESC solves the MCG problem and vice versa.

Assume we found the S-T max-flow that maximizes the number of paths containing nodes wESC. The solution for the MCG problem is composed from the sets corresponding to the nodes wESC (the nodes that receive one unit of flow from their group nodes). Note that since all C_i nodes in the same group G_j share the same ESC, then at most one of them will be wESC, i.e., only one set from each group is selected. Note also that the cardinality of the selected groups is the maximum given we select one set from each group, because otherwise the number of paths containing nodes wESC will not be the maximum too.

Now assume that we solved the MCG problem and selected the set of nodes that maximizes the cardinality of their union, such that one set from each group is selected. To find the S-T max-flow that maximizes the number of paths containing nodes wESC, we translate the

selection into flows. Simply, for each selected C_i we send flow through all of its elements, and also we send an extra unit of flow to it through the node representing its group to make it a node wESC. After this step we just augment flow on all the available S-T paths

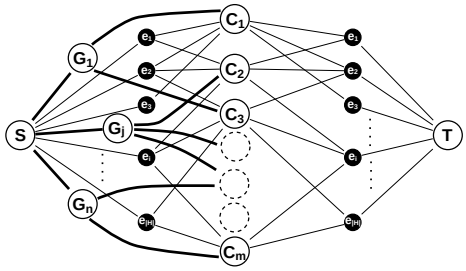


Figure 6.6 Graph resulting from reduction

□

Note that if network coding was not allowed, then from equation (6.2) we cannot protect more than ESC data units. Therefore, to utilize the extra source connectivity in a more efficient manner we should apply network coding whenever possible. Network coding can be used if a node wESC, say u, lies on more than one S-T path, and has $EC(u) \geq 1$. For example, let u be a node wESC that lies on two S-T paths, and that has $EC(u) = 1$. A network code can be designed to deliver three combinations in two data units to u, such that any two combinations are solvable, i.e., two data units are protected from S to u against a single link failure. Note that the number of failures that can be tolerated is at most $EC(u)$. Therefore, the nodes in X should have the following properties:

1. Each node $u_i \in X$ must have $f^S(u_i) > f^{u_i}(T)$.

2. The combinations received by a node $u_i \in X$ must be solvable if at most $e = f^S(u_i) - f^{u_i}(T)$ failures occurred on the $f^S(u_i)$ paths from S to u_i.

The first condition requires the flow from the source to each node $u_i \in X$ to be larger than the flow from that node to the destination. This condition is necessary to introduce redundancy in the forwarding process from S to the nodes in X. The second condition can be satisfied by designing a network code that delivers, for each node u_i, a set of $f^S(u_i)$ combinations, such

that any $f^{u_i}(T)$ combinations of them are solvable. These two conditions allow a node u_i to act as pre-cut decoding node, which can recover the data units sent from S to T through u_i, if at most $e = f^S(u_i) - f^{u_i}(T)$ failures occurred on the S-u_i link-disjoint paths, and then send these native data units to T.

In the next section we present an integer linear program (ILP) formulation of our problem. Solving the ILP will select the routes for the S-T max-flow, and will maximize the number of pre-cut-protected paths (the number of S-T paths containing nodes wESC).

6.5 Integer Linear program Formulation

We need to maximize the number of S-T paths that contain nodes wESC, regardless of the number of those nodes. We assume that the S-T max-flow equals h, and that the flow can take integer values only. Since we are interested in the number of paths containing nodes wESC, we treat each of the h units of flow as a commodity. That is, we have h commodities, each of which is responsible for selecting a single S-T path. The ILP find the routes for these h commodities on a graph with unit-capacity links, such that the number of paths containing nodes wESC is maximized. Let us begin by defining our notation:

- Let σ_i be a binary variable that equals 1 if path i (P_i) goes through at least one node wESC, and 0 otherwise. That is, $\sigma_i = 1$ if P_i is pre-cut-protected, and 0 otherwise.

- $f^i_{(a,b)}$ is the value of the flow from commodity i on link (a,b). The links forwarding f^i determines P_i.

- u^i_j is the amount of flow f^i entering node j. Although u^i_j is not constrained to be binary, it will be either 1 or 0 since the source sends only one unit of flow f^i.

- $g^j_{(a,b)}$ is the amount of extra flow g^j that is sent from the source to node j on link (a,b). A node that consumes (not forwards) this flow will be included in X.

- x_j is the amount of flow g^j entering node j. Although x_j is not constrained to be binary, it will be either 1 or 0 since the source sends only one unit of flow g^i.

- ζ_j^i is a binary variable that equals 1 if node j is on P_i and is wESC, i.e., $\zeta_j^i = u_j^i x_j$.
- d_j is the minimum hop distance of node j from the source, which is a constant that can be computed for each node before solving the ILP, e.g., using Dijkstra's shortest path algorithm.
- δ_j^i is a variable that equals d_j if $\zeta_j^i = 1$, i.e., $\delta_j^i = d_j \zeta_j^i$.
- Ω is a very large positive constant.
- w is a weighing factor for $\sum \sigma_i$, and is larger than the length of the longest possible path from the source to any node in the network, and can be set to $|E|$. This way the ILP maximizes the length of the protected paths if it does not reduce the number of protected data units.

our objective function is:

$$Maximize \quad w \sum_{i=1}^{h} \sigma_i + \sum_{\forall j} \delta_j \tag{6.3}$$

Subject to,

$$\sum_{\forall (S,b) \in E} f_{(S,b)}^i = 1, \ \forall i, \ where \ 1 \leq i \leq h \tag{6.4}$$

$$\sum_{\forall (a,b) \in E} f_{(a,b)}^i - \sum_{\forall (b,a) \in E} f_{(b,a)}^i = 0, \ \forall i, \ \forall b \in V \backslash \{S, T\} \tag{6.5}$$

$$u_j^i - \sum_{\forall (a,j) \in E} f_{(a,j)}^i = 0, \forall i, j \tag{6.6}$$

$$\sum_{\forall (S,b) \in E} g_{(S,b)}^j \leq 1, \ \forall j \tag{6.7}$$

$$x_j - \sum_{\forall (a,j) \in E} g_{(a,j)}^j = 0, \ \forall j \tag{6.8}$$

$$\sum_{\forall(k,v)\in E} g^j - \sum_{\forall(v,k)\in E} g^j = 0,\ \forall v \in V\backslash\{S,j\} \tag{6.9}$$

$$\sum_{\forall j} g^j_{(a,b)} + \sum_{\forall i} f^i_{(a,b)} \leq 1,\ \forall (a,b) \in E \tag{6.10}$$

$$\zeta^i_j - \frac{u^i_j + x_j}{2} \leq 0,\ \forall i,j \tag{6.11}$$

$$\sigma_i - \frac{\sum_{\forall j\in V} \zeta^i_j}{\Omega} < 1 \tag{6.12}$$

$$\delta^i_j - d_j\zeta^i_j = 0 \tag{6.13}$$

Constraint (6.4) forces the S-T flow to be h, and constraint (6.5) conserves all commodities on all nodes except S and T. (6.6) make $u^i_j = 1$ if node j is on path i. The extra flow that can be sent to a node wESC is bounded by 1 as shown in constraint (6.7). Constraint (6.8) sets x_j to 1 if node j receives any extra flow. The extra flow (g^j) is conserved at all nodes except the source and node j by constraint (6.9). Constraint (6.10) guarantees that the link capacity of unit of flow is not exceeded. Constraint (6.11) sets ζ^i_j to 1 if node j is on path P_i and is a node wESC. Constraint (6.12) prevents σ_i from being 1 if P_i has no node wESC. The value of δ^i_j is set to d_j if $\zeta^i_j = 1$ by constraint (6.13). Note that forcing the extra flow g^j sent to node j to be at most 1 does not affect the ILP optimality, since a path is considered pre-cut-protected if it has a node wESC regardless of the amount of extra flow received at that node.

In the next section we present a heuristic approach to solve the problem of maximizing the number of paths containing nodes wESC. Moreover, we compare the heuristic results to the results from the ILP.

6.6 Heuristic approach

Our heuristic works in three phases; the first one greedily selects an initial set X'; the second one modifies the flow on the graph (if needed) to guarantee that the S-T max-flow is achieved, and the third one utilizes any remaining connectivity and produces the final set X. The first phase works in iterations, where a single node is added to X' in each iteration. Each time we add the node that can send the most flow to the destination, while being able to receive more flow from the source, to satisfy the two conditions stated at the end of Section 6.4. If no more nodes satisfy this criteria and the S-T flow is still less than h, the second phase is entered. The second phase finds as much augmenting paths as possible from S to T so that the S-T max-flow is maximized. Finally, the third phase checks the nodes in the pre-cut portion of the graph to see if there are any remaining nodes wESC, and makes use of this extra connectivity.

6.6.1 Phase 1: Selecting the initial set X'

Recall that if all the min-cut edges are deleted, then the graph will be divided into two partitions A (pre-cut), and A' (post-cut). Note that the routing of the S-T flow in the post-cut portion of the network is independent from the routing of the S-T flow in the pre-cut portion of the network. Therefore, and since the selection of the final set X depends on the routing of the S-T max-flow in the pre-cut portion of the graph, we can simplify the graph under consideration and just focus on the sub-graph, H, induced by the nodes in A with a little modification. Specifically, given a directed graph $G(V_G, E_G)$, let $t(u, v)$ represent the tail node of edge (u,v), i.e., node u. Also, let F_S be the set of tail nodes on the min-cut edges,i.e., F_S contains the nodes in:

$$\bigcup_{i=1}^{h} t(C_i)$$

where the S-T max-flow = h, and C_i is the cutting edge on path i as defined in Section 6.2. We transform graph G to $H(V_H, E_H)$ as follows:

1. Delete the nodes in $\{V_G \backslash A\}$

2. $V_H = \{A, T'\}$, where T' is a dummy destination node.

3. $E_H = \{(u,v)|u,v \in A\} \bigcup \{(u,T')|\forall u \in F_S\}$. Note that $\{S, F_S\} \subset A$.

Each iteration of phase 1 adds the node that can send the most flow to T (or equivalently T'), while being able to receive more flow from the source. Let this node be x, then it satisfies the following conditions:

1. $f^x(T') \geq f^u(T'), \forall u \in V_H$
2. $f^S(x) > f^x(T')$

After identifying node x, the flow is sent in two steps; in the first step, $(f^x(T')+1)$ units of flow are sent from S to x, and in the second one $f^x(T')$ units of flow are sent from x to T'. This way, node x can receive redundant information to protect the $f^x(T)$ path segment from S to x. Only one extra unit of flow is sent to x so that the extra source connectivity is fairly divided between the nodes in X' at the end of phase 1.

From a network flows perspective, to forward the flow as described in the previous paragraph, $(f^x(T')+1)$ units of flow should be sent on $(f^x(T')+1)$ augmenting paths from S to x, and $f^x(T')$ units of flow should be sent on $f^x(T')$ augmenting paths from x to T'. Note that since we are working on a residual graph, the paths found from x to T' may contain backward edges, which were used initially to forward flow from S to x. If this happens then the flow sent from S to the nodes in X' may be changed and some nodes in X' may not still be nodes wESC. To resolve this issue, we can delete all the edges on the paths found from S to x in each iteration. However, this may reduce our ability to find augmenting paths from S to the nodes in H, and thus, may reduce the number of nodes that can be added to X'. Therefore, to be able to find augmenting paths without causing any of these problems we work with two copies of H. The first one, which we call H^S, is used to find paths from S to the nodes in H, and the second one, which we refer to as H^T, is used to find paths from the nodes in H to T'. The links in H^S and H^T are related to each other as follows:

- After the first step is done, and $(f^x(T')+1)$ paths were found from S to x and augmented on H^S. Every edge (u, v) in E_{H^T} that corresponds to a backward edge (v, u) in E_{H^S} is deleted.

- Similarly, after the second step is completed, and $f^x(T')$ paths were found from x to T' and augmented on H^T. Every edge (u, v) in E_{H^S} that corresponds to a backward edge (v, u) in E_{H^T} is deleted.

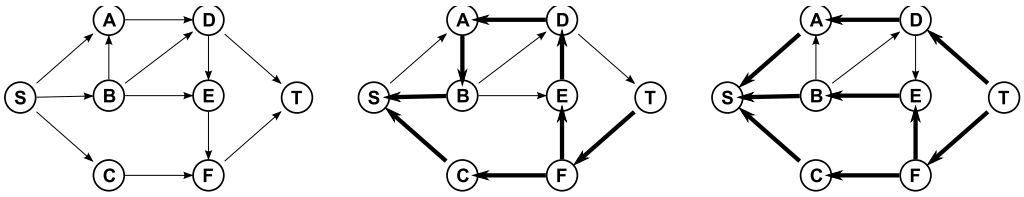
In an iteration, if two or more candidate nodes have the same flow to T', the tie is broken in favor of the largest minimum hop distance from the source, i.e., the one with the largest $d^S(u)$ is chosen to be added to X'. After that, if two or more nodes have the same flow and minimum hop distance a node is chosen randomly. Taking this into consideration, phase 1 ends when no more nodes can be added to X'.

6.6.2 Phase 2: Maximizing the S-T flow

The resulting S-T flow from phase 1 equals $\sum_{\forall x \in X'} f^x(T')$, which might be less than or equal to h (the max-flow). This is because the extra available connectivity is shared between the nodes in V_H. For example, consider the graph in Figure 6.7(a), where the S-T max-flow is 2. Phase 1 resulted in adding only one node (F) to X'. Assume that node F receives two units of flow from S along the two paths $P_1 = \{S \rightarrow C \rightarrow F\}$ and $P_2 = \{S \rightarrow B \rightarrow A \rightarrow D \rightarrow E \rightarrow F\}$, and sends one unit of flow to T on the direct edge (F, T). The resulting residual graph after augmenting these paths is shown in Figure 6.7(b), where the backward edges resulting from the augmentation process are shown in boldface. At this point, no more nodes wESC can be added to X' (because the two conditions in the previous subsection are not met for any node), but the S-T flow so far is only equal to 1. Therefore, phase 2 should be entered to maximize the S-T flow. Assume that phase 2 found the path $P_3 = \{S \rightarrow A \rightarrow B \rightarrow E \rightarrow D \rightarrow T\}$, and augmented the flow. After this step, no more S-T paths can be found on the residual graph, which means that the S-T flow is maximized, the resulting residual graph is shown in Figure 6.7(c). Note that after phase 2, node F still has two link-disjoint paths from S.

6.6.3 Phase 3: Utilizing the remaining ESC

This phase simply checks if it is possible to send extra flow to any node in H (that lies on at least one path) after the first two phases are finished. If a node u is found to be able to receive



(a) Graph H with S-T max-flow = 2

(b) Residual graph after phase 1

(c) Residual graph after phase 2

Figure 6.7 The operation of the first two phases. (b) node F is added to X', and it receives two units of flow from S and send one unit of flow to T. (c) The S-T max-flow is maximized

extra flow e' from S, then if it is not already in X' it should be added to X'. The number of data units node u sends to T' equals $f^u(T')$. The number of data units or combinations it can receive from S is $k = f^u(T') + e'$ if it is not in X', and is $k = f^u(T') + e' + 1$ if it is already in X'. If $f^u(T') = 1$ no coding is needed and we need to just send copies of the same forwarded data unit on all the paths to u. However, if $f^u(T') > 1$ a network code should be designed to deliver k combinations to u such that any $f^x(T)$ of them are solvable. Algorithm 9 summarizes the three phases.

6.6.4 Evaluation

In this section we compare the results from our heuristic to the results from the ILP presented in Section 6.5. The heuristic was compared to the ILP in five different cases. Each case represents a different network size, where the number of network nodes V was changed to take the values $\{5, 10, 15, 20, 25\}$. In each case eighty random network instances were generated, and fed to the heuristic and the ILP. Figure 6.8 shows the ratio between the average number of protected paths by the heuristic and the average number of protected paths by the ILP for the eighty runs. The figure shows that the performance of our heuristic is acceptable, where in the worst case at $V = 20$ it was around 77% of the optimal on average.

To gain a better insight on the operation of the heuristic compared to the ILP we measured

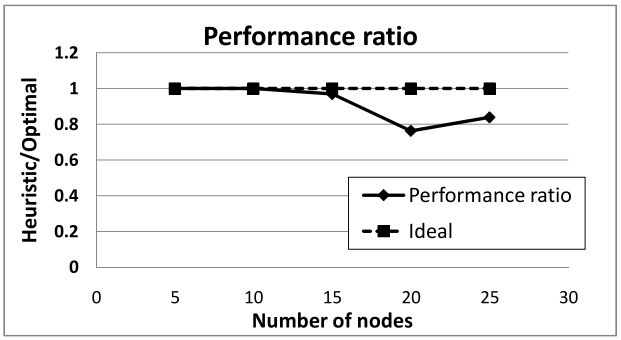


Figure 6.8 Ratio of the number of protected paths by the heuristic to that of the ILP for different number of nodes

the S-T max-flow, counted the number of pre-cut-protected paths from the heuristic, and the number of pre-cut-protected paths resulting from the ILP in each time the heuristic and the ILP were executed (on the same network instance).

The histograms for the cases of V = 10, 15, 20, and 25 are shown in Figures 6.9(a), 6.9(b), 6.9(c), and 6.9(d) respectively. In general, the results from the heuristic are close to those from the ILP. Note that in some cases, the number of times the heuristic is able to protect X_1 paths may be larger than the number of times the ILP is able to protect the same number of paths X_1. However, this does not invalidate the heuristic because it comes at the price of protecting a larger number of paths $X_2 > X_1$ a fewer number of times. For example, in Figure 6.9(c), the heuristic was able to protect $X_1 = 2$ paths more than the ILP, but the ILP was able to protect $X_2 = 4$ paths more than the heuristic.

6.6.5 Coding

The resulting S-T' flow from the heuristic (or the ILP) can be decomposed into two parts; the first, a one-to-many flow from S to the nodes in X, and the second is a many-to-one flow from S and the nodes in X to T'. The many-to-one flow is not and cannot be coded, since it is composed from the h native data units that are forwarded from S and the nodes in X (possibly after decoding) to T', on h disjoint paths. However, the one-to-many flow from S to the nodes in X can, and should be coded to utilize the extra source connectivity in the most efficient manner. Note that this one-to-many flow is different from normal multicast flow since

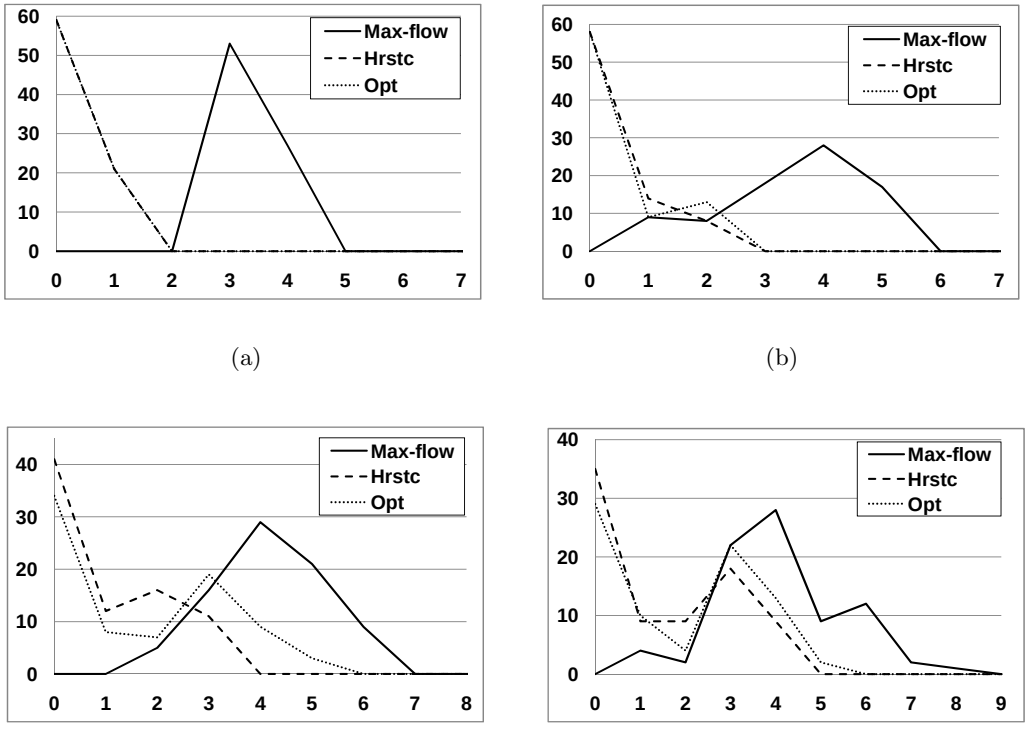


Figure 6.9 All figures are histograms, which count three different frequencies: the max-flow, the number of protected paths from the heuristic and the number of protected paths from the ILP. (a) has $V = 10$, (b) has $V = 15$, (c) has $V = 20$ and (d) has $V = 25$. The x axis is the number paths either protected or counted in the max-flow, and the y axis is the number of times each number of paths occurred as a max-flow or protected by the ILP or the Heuristic

different data is sent to different nodes. Therefore, a standard multicast network code cannot be used. In fact, the coding in our case is simpler, and needs to be done at a limited number of network nodes as we will show in the following discussion.

After the heuristic is done and the flow is constructed in the pre-cut portion of the graph. A node $u \in X$ can receive $k + e = Flow_S[u]$ units of flow from S and can send $k = Flow_T[u]$ units of flow to T' (these values were computed in the heuristic). This implies that there are $k + e$ edge-disjoint paths from S to u, and k edge disjoint paths from u to T' (or equivalently to T). Note that k represents the number of S-T paths (or data units) going through node u, and that e represents the paths used to carry redundant information to u.

Let N_{x_i} be the set of 1-hop neighbors of the source on all the $k + e$ paths from S to x_i. Assume that all the nodes in N_{x_i} have received the same set of k data units from the source (the k data units on the k S-T paths). To construct a network code that delivers $k+e$ combinations to x_i such that any k of them are solvable using the received data units, we need to assign the proper coding vectors to the nodes in N_{x_i}. The coding vectors can be assigned from an $k \times (k+e)$ matrix that has no singular $k \times k$ submatrices, i.e., any $k \times k$ submatrix is invertible. A class of matrices that satisfies this requirement is the Cauchy matrices (29). Therefore, we can simply assign to each node in N_{x_i} a column from a $k \times (k + e)$ Cauchy matrix, such that no two nodes are assigned the same column.

However, such a coding scheme requires decoding at the nodes in X in each transmission round. An alternative way that will require a fewer number of decoding operations would be to use a systematic code. In a systematic code, k out of the $k + e$ combinations will be trivial combinations, where each of which carries one of the k native data units. In this case, decoding is necessary at a node $x_i \in X$, only if one of the native data units was lost due to a failure on one of the k S-T paths going through node x_i. A simple way to do this is presented in (30). Basically, let $\mathcal{M}_i$ denote a $k \times (k + e)$ Cauchy matrix with columns representing the coding vectors of the nodes in N_{x_i}. We can view $\mathcal{M}_i$ as two side-by-side matrices $\mathcal{M}_i = (\mathcal{M}_{k_i}|\mathcal{M}_{e_i})$, where $\mathcal{M}_{k_i}$ is a $k \times k$ matrix , and $\mathcal{M}_{e_i}$ is a $k \times e$ matrix as follows:

$$\mathcal{M}_i = \left[\begin{array}{ccc|ccc} \alpha_{0,0} & \dots & \alpha_{0,k-1} & \alpha_{0,k} & \dots & \alpha_{0,k+e-1} \\ \alpha_{1,0} & \dots & \alpha_{1,k-1} & \alpha_{1,k} & \dots & \alpha_{1,k+e-1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{k-1,0} & \dots & \alpha_{k-1,k-1} & \alpha_{k-1,k} & \dots & \alpha_{k-1,k+e-1} \end{array}\right]$$

Let $\mathcal{M}'_i$ be the $k \times (k+e)$ matrix resulting from multiplying $\mathcal{M}_{k_i}^{-1}$ by $\mathcal{M}_i$:

$$\mathcal{M}'_i = \mathcal{M}_{k_i}^{-1} \times \mathcal{M}_i = (I_k | \mathcal{M}_{k_i}^{-1} \times \mathcal{M}_{e_i}) = (I_k | \mathcal{M}'_{e_i}) =$$

$$\left[\begin{array}{cccc|ccc} 1 & 0 & \dots & 0 & \alpha'_{0,k} & \dots & \alpha'_{0,k+e-1} \\ 0 & 1 & \dots & 0 & \alpha'_{1,k} & \dots & \alpha'_{1,k+e-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & \alpha'_{k-1,k} & \dots & \alpha'_{k-1,k+e-1} \end{array}\right]$$

Since the original matrix $\mathcal{M}_i$ has no singular submatrices, then the resulting matrix $\mathcal{M}'_i$ has no singular submatrices also. Note that although the non-singularity property is preserved, the matrix is no longer a Cauchy matrix. Therefore, given that the source has already transmitted the k data units to the nodes in N_{x_i}, assigning the columns of $\mathcal{M}'_i$ to the nodes in N_{x_i} will create $k+e$ combinations such that any k of them are solvable. Moreover, the code is systematic, where out of the $k+e$ combinations there are k trivial combinations, each of which is composed of a single native data unit.

A special case is when $e = 1$. In this case, after the source finishes transmitting the k data units to the nodes in N_{x_i} (where $|N_{x_i}| = k+1$), one of the nodes in N_{x_i} can sum all the received data units and send this sum along with the k native data units on $k+1$ paths to x_i.

6.7 Post-Cut: Nodes with Extra Destination Connectivity

Nodes with extra destination connectivity (wEDC) can be found in the post-cut portion of the network only. Nodes wEDC (or a subset of them) can act as post-cut encoding nodes, which create and send redundant combinations to the destination node T to enhance the survivability

of the information flow. Note that this case is different from the one considered previously, because all the nodes wEDC are sending their data units to the same destination. Let $hd(u, v)$ denote the head node of edge (u, v), and let F_T be the closest set of nodes wEDC to S, or equivalently the farthest set of node wEDC from T, then F_T contains the nodes in:

$$\bigcup_{i=1}^{h} hd(C_i)$$

where C_i is the cutting edge on path P_i. Note that if an edge C_i has T as a head node, then $T \in F_T$, which means that the data unit on the cutting edge C_i is delivered to the destination directly after the cut and cannot be protected. That is, the flow that can be protected from the nodes in F_T is reduced by the number of edges in the cut incident to T. Let $F'_T = F_T \backslash T$, note that $0 \leq |F'_T| \leq h$ (0 when all the nodes in F_T are direct neighbors to T, and h when none of them is a direct neighbor to T), where $h = f^S(T)$. Also, note that since the nodes in F'_T are the head nodes of edges in the min-cut, then we have $f^S(F'_T) = |F'_T|$. That is, each node in F'_T has only one data unit to forward to T, and $|F'_T|$ is the maximum post-cut flow that can be protected. Let e denote the total available extra destination connectivity from the nodes in F'_T, then e is calculated as follows

$$e = f^{F'_T}(T) - |F'_T|.$$

Note that if $F'_T \neq \emptyset$, then $e \geq 1$. If network coding is not allowed, then no more than e data units can be protected. However, if network coding is allowed, we prove that all the data units in F'_T can be protected against at least a single failure:

Theorem 8. *Let F'_T be the set of head nodes of the closest min-cut edges to T, such that $T \notin F'_T$. Then if network coding is allowed, the data units at the nodes in F'_T can be protected together against a single failure.*

Proof. It was shown in Chapter 3 that a many-to-one flow, similar to the flow from F'_T to T, can be protected against a single link failure (using network coding) if and only if any subset of k source nodes can reach the common destination node through at least $k + 1$ edge-disjoint

paths.

Therefore, to prove the theorem we need to prove that any k nodes in F'_T can reach T through at least $k+1$ edge-disjoint paths. We prove this by contradiction. Assume that there is a set, Q, of k nodes in F'_T that can reach T through only k edge-disjoint paths, i.e., $f^Q(T) = k$. Then there are k cutting edges on the k paths from the nodes in Q to T, which contradicts the assumption of the single min-cut. Therefore, any k nodes in F'_T must be able to reach the destination node T through at least $k+1$ link disjoint paths, which concludes the proof. □

If $e = 1$ we can use the coding tree approach presented in Chapter 3. However, if $e > 1$, then to be able to recover the $|F'_T|$ data units if at most e failures occurred in the post-cut portion of the graph, we need two conditions to be satisfied. First, any set of k nodes in F'_T must be able to reach the destination through at least $k+e$ link-disjoint paths. Second, we need to assign coding vectors to the $f^{F'_T}(T)$ combinations such that any $|F'_T|$ vectors from them are linearly independent. Note that if $e > 1$, then the linear independence of any $|F'_T|$ vectors does not necessarily mean that we can recover the $|F'_T|$ data units from any $|F'_T|$ combinations. This is because, when e is larger than 1, the first condition is not necessarily satisfied. To clarify this issue, consider the example in Figure 6.10. In the figure, $F'_T = \{A, B, C, D\}$, $f^{F'_T}(T) = 6$ and $e = 2$. The black nodes represent the 6 paths from F'_T to T, and c_i is the combination carried on P_i. The links represent the ability of the nodes in F'_T to reach the different paths. If a path P_i can be reached by k nodes in F'_T then c_i is a function of k data units. Note that since $e = 2$, the first condition stated above is not satisfied, because nodes C and D can reach T through only three paths not four. To satisfy the second condition, the coding vectors can be chosen as the columns of a 4×6 Cauchy matrix. Now consider the four combinations $\{c_1, c_2, c_3, c_4\}$. Since c_1, c_2 and c_3 are functions of only two data units A and B (i.e., the coefficients of C and D are zeros), then the three combinations are linearly dependent. However, note that any two combinations of them are linearly independent, because in a Cauchy matrix any square submatrix has full rank (since it is another Cauchy matrix). That is, although the four combinations are in four data units (because of c_4), only three are linearly independent and

only two are solvable.

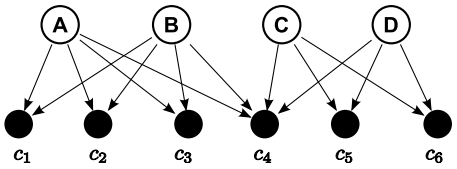


Figure 6.10 $F'_T = \{A, B, C, D\}$, $f^{F'_T}(T) = 6$, and $e = 2$. The combinations c_1, c_2, and c_3 are functions of A and B. The combinations c_5 and c_6 are functions of C and D. Combination c_4 is a function of A, B, C and D. The set $Q = \{c_1, c_2, c_3, c_4\}$ has 3 linearly independent combinations, from which only two can be solved to recover A and B

If each node $v \in F'_T$ has $f^v(T)$ paths to T that are link disjoint from the paths from all other nodes in F'_T to T, network coding will not be necessary and each node in F'_T can send $f^v(T)$ copies of its data on its $f^v(T)$ paths to T. However, network coding becomes necessary if the paths from the nodes in F'_T to T share links. The first links to be shared are in the link-cut between F'_T and T that is closest to F'_T.

Let $f^{F'_T}(T) = n$, then there are n edge-disjoint paths $\{P'_1, \ldots, P'_n\}$ from F'_T to T. Let C'_i denote the cutting edge on path P_i from a node in F'_T to T that is closest to F'_T (if path P'_i has more than one cutting edge). Recall that C'_i is a cutting edge only if the maximum achievable F'_T-T flow is reduced by 1. Let Z be the set of coding nodes, which contains the tail nodes of all the n cutting edges in $\bigcup_{i=1}^{n} C'_i$. Note that $|Z| \leq n$, and that network coding is not necessary at any of the downstream nodes after Z, since the combinations created at the nodes Z will be forwarded to T on $|F'_T|$ edge-disjoint paths. Let $|F'_T| = m$, then a network code can be constructed by assigning each edge C'_i, where $1 \leq i \leq n$, a distinct column from an $m \times n$ Cauchy matrix. The solvability of any m combinations depends on how the nodes in F'_T are connected to T as shown in the previous example. Specifically, let r be the minimum number of solvable combinations in any m combinations, and let q denote the number of failures in the post-cut portion of the graph. Then we are guaranteed the full recovery of the m data units if $q = 1$ (by Theorem 2), and we are guaranteed the partial recovery of at least r data units if $q = e$ (by the definition of r).

6.8 Summary

We presented a new protection approach, called max-flow protection, which can enhance the survivability of the whole S-T max-flow. The basic idea is not to protect links in the min-cut, but try to protect all other links if possible. We divided the problem into two problems; pre-cut protection and post-cut protection. Pre-cut protection is NP-hard. Therefore, the problem is formulated as an ILP, and a heuristic is proposed to solve it. We showed that all data units that are not delivered directly to T after the min-cut can be post-cut-protected. Finally, simple network codes are proposed to maximize the number of pre- and post-cut protected paths.

Algorithm 9 Selecting set X

Input: Graph $H(V_H, E_H)$, h = S-T max-flow
Output: Set X containing nodes wESC

$X' = \phi$, $ST\_flow = 0$, $Phase\_done = 0$
Create matrices $Flow_S[V_H]$, $Flow_T[V_H]$ //One dimensional matrices initialized to all zeros, to store the final flow from S to each node in X, and from each node in X to T'. This information will be used for coding later
//Phase 1
Create graphs H^S and H^T, where $V_{H^S} = V_{H^T} = V_H$ and $E_{H^S} = E_{H^T} = E_H$.
while $(Phase\_done == 0)$ **do**
  Compute $f^S(u)$ on graph H^S, $\forall u \in V_{H^S}$
  Compute $f^u(T')$ on graph H^T, $\forall u \in V_{H^T}$
  Select node x, where $f^x(T') \geq f^u(T'), \forall u \in V_H$, and $f^S(x) > f^x(T')$
  if (No such node exists) **then**
    $Phase\_done = 1$
  else
    Find $f^x(T') + 1$ augmenting paths from S to x on H^S
    Delete all forward edges in H^T if they are reversed in H^S//due to augmentation
    Find $f^x(T')$ augmenting paths from x to T'
    Delete all forward edges in H^S if the are reversed in H^T
    $X' = X' \cup \{x\}$
    $ST\_flow = ST\_flow + f^x(T')$
    $Flow_S[x] = f^x(T') + 1$
    $Flow_T[x] = f^x(T')$
  end if
end while
for all $((u, v) \in E_H)$ **do**
  if $((v, u) \in E_{H^S} || (v, u) \in E_{H^T})$ **then**
    Reverse (u, v) in H
  end if
end for
$Phase\_done = 0$ //End of Phase 1, and beginning of Phase 2
while $(Phase\_done == 0)$ **do**
  if $(ST\_flow = h)$ **then**
    $Phase\_done = 1$
  else
    Find an S-T' augmenting path in H
    ST_flow++
  end if
end while
$Phase\_done = 0$ //End of Phase 2, and beginning of Phase 3
for all $(u \in V_H)$ **do**
  Compute $p = f^S(u)$ on the current residual graph of H
  if $(f^S(u) > 0)$ **then**
    Find p augmenting paths from S to u on H
    $Flow_S[u] = Flow_S[u] + p$
  end if
  if $(u \notin X')$ **then**
    Compute $q = f^{T'}(u)$ on H^T
    $Flow_T[u] = Flow_T[u] + q$
    $X' = X' \cup \{u\}$
  end if
end for
return X'

CHAPTER 7. Summary and future work

7.1 Summary

This dissertation focused on enhancing the survivability of information flow in multi-hop wireless networks (MWN). We focused on the case of many-to-one flow since it is used in two important types of MWNs (from the routers to the gateway in WMNs, and from the sensors to the base station in WSNs). First, we presented a centralized approach that enables the data flows from n sources to be protected together against single link-failures using only $n+1$ paths. To do this, we showed how to design a proper network code using only {0,1} coefficients, which allows the creation of $n+1$ combinations, such that, any n of them are solvable. We provided a detailed study of this technique, which covered extensions for more general cases, complexity analysis, and performance evaluation that showed the efficiency of the proposed approach. In addition, we studied the problem of scheduling the sources transmissions in such a many-to-one flow. We studied two version of the problem; the first assumes that normal digital network coding (DNC) is used, and the second assumes analog network coding (ANC) is used. We showed that, theoretically, ANC-based scheduling can outperform DNC-based scheduling by at most a factor of n (the number of sources). However, in practice, our simulation showed that the gain of ANC is usually around 2.

Second, we presented a simple and efficient technique that uses deterministic binary network coding in a distributed manner to enhance the survivability of the sensors-to-base information flow against packet loss. We extended this scheme to cover multiple losses, and to cover link or node failures. We showed that the overhead of forwarding the coding vectors can be reduced using relative indexing, and we showed that the coding scheme allows partial recovery of data units, if more than one failure takes place, given that some conditions are satisfied.

Furthermore, we showed through a simulation study, using the TOSSIM simulator, that our scheme is highly scalable and performs better as the number of sources increases.

Finally, we studied the survivability of a unicast communication session, where a single source node S is transmitting data to a single destination node T through multiple hops. We presented max-flow protection, which is a different way to handle the "survivability vs. bandwidth" tradeoff. We showed how to enhance the survivability of the S-T information flow without reducing the maximum achievable S-T information rate, which comes at the cost of not protecting min-cut. We divided the max-flow protection problem into two sub-problems; pre-cut protection, and post-cut protection. We proved the hardness of pre-cut protection, and presented an ILP and a heuristic approach to solve it. For post-cut protection, we proved that all the data units that are not delivered to T directly after the min-cut can be protected against a single link failure.

7.2 Future work

Our studies showed that network coding can significantly reduce the used network resources for protection purposes, and still achieve recovery speeds comparable to that of traditional protection schemes. Also, we showed that given a limited amount of network resources (e.g., the extra connectivity in Chapter 6) network coding can maximize the number of protected data units. In both cases, network coding proved to be useful and advantageous in efficiently utilizing the available network resources to provide proactive protection. In our future work, we plan to study the following problems:

- The max-flow protection approach was introduced for a single unicast session. We plan to investigate possible extensions of the max-flow protection approach for other communication modes like the many-to-many group communication mode, or for the case of multiple unicast sessions.

- The relatively new cognitive radio networks (CRNs) present a very challenging environment for developing survivability mechanisms. Most of the work done in CRNs deals

with channel sensing and channel allocation, even the routing problem by itself is still not well investigated. Since channel access is limited, and may be interrupted anytime by primary channel users, a survivable routing protocol is crucial for such networks.

- In our work we have shown how to perform network coding using a binary field, i.e., the coefficients are in $\{0, 1\}$. This was enough to tolerate a single failure. However, if we want to tolerate multiple failures a binary field may not always be sufficient. In our future work, we plan to investigate the relation between the number of failures (to be tolerated) and the needed field size.

APPENDIX A. Finding Bipartite equivalent

Creating a feasible set of combinations at the nodes in L_s depends on the data units that can reach these nodes from the sources in U_s. The assumption of a bipartite induced graph in Section 3.3.1 accurately captures the data unit reachability to the nodes in L_s. Therefor, if the original graph induced by the nodes in U_s and L_s was not bipartite we need to find a bipartite equivalent.

As mentioned earlier, user nodes do not communicate directly in practice, rather, they use the infrastructure formed by the routers to communicate. Therefore, there is only one case in which the induced graph is not bipartite, and that is when there is a link between two routers in L_s. Such a graph is called semi-bipartite since only one of its partitions can contain both ends of a link. An example is shown in Fig. A.1. In a semi-bipartite graph the links from U_s to L_s nodes are not enough to accurately capture the reachability of data units to the routers. For example in Fig. A.1 S_1 can reach router C through router node B, but there is no direct link between S_1 and C.

In this section we introduce a simple transformation to find the bipartite equivalent of any semi-bipartite graph, in terms of data unit reachability in a certain level i. Assume that the edge (R_i, R_j) connects router R_i with router R_j, then to remove (R_i, R_j) we add the following logical edges:

1. Add edge (u, R_j), if it does not already exist, for all u such that $(u, R_i) \in E$.

2. Add edge (u, R_i), if it does not already exist, for all u such that $(u, R_j) \in E$.

The transformation of the semi-bipartite subgraph shown in Fig. A.1 is illustrated in Fig. A.2. Where we represent the ability of users to reach routers through other router nodes by

direct links. For example, S_1 is now connected directly to router C. It is worth mentioning that this transformation is not necessary to find a feasible set of combinations if the condition of Lemma 1 is already satisfied. That is, we can just ignore the links between the routers. However, the transformation may help in reducing the needed number of combinations by adding the logical links. For example, the condition in Lemma 1 is satisfied in the network in Fig. A.3. Thus, we can ignore the link (D, E), and in this case we need n+2 combination to be able to tolerate a single link failure. However, if the bipartite equivalent is used, only n+1 are needed as shown in Fig. A.4. Therefore, it is better to do the transformation whenever the induced graph by the nodes in U_s and L_s is semi-bipartite, since this will allow us to maximize the throughput and reduce the used resources.

Using this transformation, we can always assume that the induced graph by the nodes in U_s and L_s is bipartite. In the next subsection we discuss how this transformation works with the coding and scheduling mechanisms introduced in Sections 3.3 and 4.

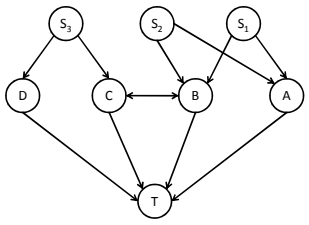


Figure A.1 A semi-bipartite graph

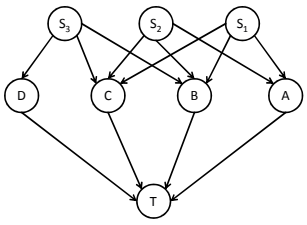


Figure A.2 Transformed graph

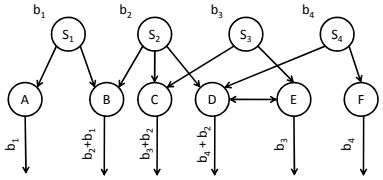


Figure A.3 need n+2 combinations if transformation is ignored

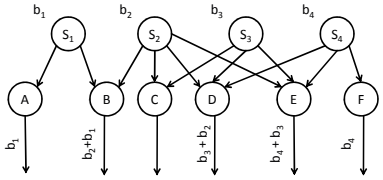


Figure A.4 need n+1 combinations if transformation is done

APPENDIX B. The effect of transformation on coding and scheduling

We briefly discuss how the transformation introduced in the previous appendix works with the coding technique presented in Section 3.3, and the scheduling mechanisms presented in Chapter 4.

The coding tree algorithm tells each node in L_s how to incorporate data units in the combination originating from it. This is simply known from the tree structure found by the algorithm, where the edges are assigned a coefficient of 1 if they are on the tree and 0 otherwise. It is important to notice that this only works if all the neighbors of a node in L_s were in U_s. Otherwise, if the tree included a link between two nodes in L_s (in a semi-bipartite graph) these two nodes will not be able to correctly decide on the data units to be included. Therefore, the transformation is a must for the coding tree to run properly.

Unlike the coding tree, the scheduling algorithm introduced in Section 4 must not use the transformation and must be executed over the original graph to construct the proper schedule. This is because if the transformed graph is used, then the newly added edges will incorrectly increase the contention between sources. For example, in the graph shown in Fig. A.1 the transmission of source S_1 does not reach router C directly, and thus, if the scheduling algorithm used the bipartite equivalent in Fig. A.2 it will incorrectly consider C as a neighbor to S_1. Therefore, scheduling must be done using the original graph.

BIBLIOGRAPHY

[1] R. Diestel. Graph theory (electronic edition 2005). Springer-Verlag Heidelberg, New York 1997, 2000, 2005.

[2] A. E. Kamal. 1+n network protection for mesh networks: Network coding-based protection using p-cycles. To appear in IEEE/ACM Trans on. Net.

[3] S. Katti, S. Gollakota, and D. Katabi. Embracing wireless interference: Analog network coding. In the proceedings of SIGCOMM 2007.

[4] I. F. Akyildiz, X. Wang, W. Wang. Wireless mesh networks: a survey. Computer Networks 47 (2005) 445Ű487.

[5] S. R. Li, R. W. Yeung, and N. Cai. Linear network coding. IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 49, NO. 2, FEBRUARY 2003.

[6] R. Koetter and M. Médard. An algebraic approach to network coding. IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 11, NO. 5, OCTOBER 2003.

[7] C. Fragouli and E. Soljanin. Information flow decomposition for network coding. IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 52, NO. 3, MARCH 2006.

[8] S. Jaggi P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial time algorithms for multicast network code construction. IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 51, NO. 6, JUNE 2005.

[9] C. Fragouli, D. Katabi, A. Markopoulou, M. Mt'edard, and H. Rahul. Wireless network coding: Opportunities & challenges. In Proceedings of MILCOM 2007.

[10] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream control transmission protocol. The Internet Engineering Task Force (IETF), http://www.ietf.org/rfc/rfc2960.txt, Oct 2000.

[11] D. S. Lun, M. Medard, and M. Effros. On coding for reliable communication over packet networks. In the Proceedings of the 42nd Annual Allerton Conference on Communication, Control, and Computing (Allerton 2004), Sept./Oct. 2004, invited paper.

[12] D. Johnson, D. Maltz, and J. Broch. Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks. http://www.monarch.cs.rice.edu/monarch-papers/dsr-chapter00.pdf.

[13] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing. The Internet Engineering Task Force (IETF), http://www.ietf.org/rfc/rfc3561.txt, Jul 2003.

[14] V. Sharma and F. Hellstrand. Framework for multi-protocol label switching (mpls)-based recovery. The Internet Engineering Task Force (IETF), http://www.ietf.org/rfc/rfc3469.txt, Feb 2003.

[15] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. Computer Networks 38 (2002) 393Ű422.

[16] F. Ye, G. Zhong, S. Lu, and L. Zhang. Gradient broadcast: A robust data delivery protocol for large scale sensor networks. ACM Wireless Networks (WINET), vol. 11, no. 2, March 2005.

[17] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. ACM SIGMOBILE Mobile Computing and Communications Review archive Volume 5 , Issue 4 (October 2001).

[18] W. Lou. An efficient n-to-1 multipath routing protocol in wireless sensor networks. In Proc. 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2005).

[19] J.L. Bredin, E.D. Demaine, M. Hajiaghayi, and D. Rus. Deploying sensor networks with guaranteed capacity and fault tolerance. In Proc. 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc 2005).

[20] Y. Sankarasubramaniam O.B. Akan I.F. Akyildiz. Esrt: Event-to-sink reliable transport in wireless sensor networks. In Proc. 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc 2003).

[21] S. Kim, R. Fonseca, and D. Culler. Reliable transfer on wireless sensor networks. In Proc. SECON 2004.

[22] S. Dulman, T. Nieberg, J. Wu, and P. Havinga. Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks. In Proc. WCNC 2003.

[23] R. W. Yeung, S. R. Li, N. Cai, and Z. Zhang. *Network Coding Theory.* now Publishers Inc, 2006.

[24] Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi. A framework for reliable routing in mobile ad hoc networks. In Proceedings of Infocom 2003.

[25] M. K. Marina and S. R. Das. On-demand multipath distance vector routing in ad hoc networks. In Proceedings of ICNP 2001.

[26] R. Ahlswede, N. Cai, S. R. Li, and R. Yeung. Network information flow. Information Theory, IEEE Transactions on. Volume 46, Issue 4, July 2000 Page(s):1204 - 1216.

[27] A. Sprintson, S.Y.E. Rouayheb, and C.N Georghiades. Robust network coding for bidirected networks. Information Theory and Applications Workshop, 2007.

[28] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In proceedings of APPROX 2004, Lecture Notes in Computer Science 3122, 72Ű83.

[29] F. J. MacWilliams and N. J. A. Sloane. The theory of error-correcting codes. North Holland, 1977.

[30] J. Lacan and J. Fimes. A construction of matrices with no singular square submatrices. In proceedings of the 7th International Conference on Finite Fields and Applications, May 2003.

[31] A. Srinivas and E. Modiano. Minimum energy disjoint path routing in wireless ad-hoc networks. In the proceedings of the 9th annual international conference on Mobile computing and networking (Mobicom 2003). Pages: 122 - 133.

[32] J. Tang, G. Xue, and W. Zhang. Energy efficient survivable broadcasting and multicasting in wireless ad hoc networks. In the proceedings of Military Communications Conference, 2004 (MILCOM 2004).Pages:1165 - 1171 Vol. 3.

[33] P. Thulasiraman, S. Ramasubramanian, and M. Krunz. Disjoint multipath routing to two distinct drains in a multi-drain sensor network. In Proceedings of Infocom 2007.

[34] S. Lee and M.Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In Proceedings of ICC 2001.

[35] G. Gupta and M. Younis. Fault-tolerant clustering of wireless sensor networks. Wireless Communications and Networking, 2003. WCNC 2003.

[36] M. Virendra, S. Upadhyaya, V. Kumar, and V. Anand. Sawan: A survivable architecture for wireless lans. In Proceedings of the Third IEEE International Workshop on Information Assurance,IWIA 2005.

[37] L. Kant and W. Chen. Service survivability in wireless networks via multi-layer self-healing. In the proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2005). Page(s):2446 - 2452 Vol. 4.

[38] C. Sengul and R. Kravets. Bypass routing: An on-demand local recovery protocol for ad hoc networks. Elsevier Ad Hoc Networks Journal, Volume 4, Number 3, pp. 380-397, 2006.

[39] A. Valera, W.K.G. Seah, and S. Rao. Cooperative packet caching and shortest multipath routing in mobile ad hoc networks. In Proceedings of INFOCOM 2003.

[40] R. Leung, J. Liu, E. Poon, A. C. Chan, and B. Li. Mp-dsr: A qos-aware multi-path dynamic source routing protocol for wireless ad-hoc networks. In Proceedings of LCN 2001.

[41] W. Choi, S.K. Das, and K. Basu. Angle-based dynamic path construction for route load balancing in wireless sensor networks. In Proceedings of WCNC 2004.

[42] N. Li and J. C. Hou. Flss: A fault-tolerant topology control algorithm for wireless networks. In the proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom 2004). Pages: 275 - 286.

[43] Y. Wu, F. Wang, M. T. Thai, and Y. Li. Constructing k-connected m-dominating sets in wireless sensor networks. In Proceedings of MILCOM 2007.

[44] G. Parissidis, V. Lenders, M. May, and B. Plattner. Multi-path routing protocols in wireless mobile ad hoc networks: A quantitative comparison. NEW2AN 2006, LNCS 4003, pp. 313 - 326, 2006.

[45] P. Rogers and N. Abu-Ghazaleh. *Algorithms and Protocols for Wireless, Mobile Ad Hoc Networks*, chapter 3. Wiley-IEEE Press, November 10 2008.

[46] D. Zhou and S. Subramaniam. Survivability in optical networks. IEEE Network, vol. 14, pp. 16Ű23, Nov./Dec. 2000.

[47] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Mt'edard, and J. Crowcroft. Xors in the air: Practical wireless network coding. In the proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2006). Pages: 243 - 254.

[48] J. Kleinberg and E. Tardos. *Algorithm Design.* Addison Wesley, 2005.

[49] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows.* Prentice Hall, 1993.

[50] P. C. Li and M. Toulouse. Variations of the maximum leaf spanning tree problem for bipartite graphs. Information Processing Letters archive. Volume 97 , Issue 4 (February 2006). Pages: 129 - 132.